INTELLIGENT IN-VEHICLE SAFETY AND SECURITY MONITORING SYSTEM

by

Xiaodi Fu, MS

THESIS

Presented to the Faculty of

The University of Houston-Clear Lake

In Partial Fulfillment

Of the Requirements

For the Degree

MASTER OF SCIENCE

in Computer Information Systems

THE UNIVERSITY OF HOUSTON-CLEAR LAKE

DECEMBER, 2019

INTELLIGENT IN-VEHICLE SAFETY AND SECURITY MONITORING SYSTEM

by

Xiaodi Fu

APPROVED BY

_____

Jiang Lu, PhD, Chair

_____

Wei Wei, PhD, Committee Member

_____

Xiaokun Yang, PhD, Committee Member

RECEIVED/APPROVED BY THE COLLEGE OF SCIENCE AND ENGINEERING:

_____

David Garrison, PhD, Associate Dean

_____

Miguel A. Gonzalez, PhD, Dean

Dedication

I would love to dedicate my thesis to my husband and daughter. Without understanding and help from my husband, it is impossible for me to attend school in America. His encouragement makes me never give up in tough times. Because of studying, I do not have enough time to accompany my daughter in activities. Thanks for her understanding. They are treasure in my life.

Acknowledgments

Before anything else, I would first like to thank my committee chair, Dr. Jang Lu for his perseverant guidance, patience, and support throughout the course of the research. Without his great help, this research would not have been possible.

Also, thanks to Dr. Wei Wei and Dr. Xiaokun Yang for their availability and patience as my committee members, and for their valuable and insightful advice on the research.

Thanks also to all friends, colleagues, the department faculty and staff for making my time at the University of Houston – Clear Lake an invaluable experience. Finally, thanks to my father, mother, and all my family members for their love and encouragement.

ABSTRACT

INTELLIGENT IN-VEHICLE SAFETY AND SECURITY MONITORING SYSTEM

Xiaodi Fu
University of Houston-Clear Lake, 2019

Chair: Jiang Lu, PhD

Dangerous situations such as children are left in vehicles, are dropped off at wrong stops, or take on wrong school buses usually caused by the negligence of drivers. This paper presents a real-time intelligent in-vehicle monitoring system that can count and recognize people as well as alert drivers if such improprieties or potential dangers happen. The system uses an HOG-based face detector from the Dlib library to obtain face counting function. Face recognition is achieved through two steps, facial feature extraction, and face identification. The ResNet is used in facial feature extraction. It transforms an aligned face into a 256-dimensional vector, a Euclidean facial embedding. In face identification, labeled faces will be transformed into facial embeddings first. Then k-nearest neighbor classifier (kNN) is adopted to identify people using such facial embeddings. The simulation on ChokePoint dataset is tested and the average accuracy is 93 percent. The distance sensor performs well when it is installed 100 cm in front of people. Whether the motion sensor is installed depends on special conditions.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER I:

INTRODUCTION

1.1 Background

At present, the security of children on a school bus is an important issue that parents and teachers are concerned about. Most accidents occur among children in kindergarten and grade one because of drivers' neglect. The issue comprises of several aspects. For instance, children are dropped off at wrong bus stops [1][2][3]. Some children may forget their route numbers that lead to them getting on wrong buses [4]. It is not news that children are left in school buses because drivers do not give enough attention to them. That type of accidents, in general, happens when the age of children between 4 to 7. The situation is very dangerous when weather is bad. 37 young children, on average, died on account of vehicular heatstroke every year in America [5]. In a report about 700 child vehicular heatstroke deaths from 1998 to 2017, 54% of children died because drivers forgot them. 28% of deaths occurred for children playing in unattended vehicles and 17% of children were intentionally left in vehicles by drivers. Therefore, it is important to take effective measures to protect children from this kind of accident.

To avoid tragedies or decreasing the probability of tragedies, we present an intelligent in-vehicle safety and security system based on Raspberry Pi, Pi camera, distance sensor, and motion sensor. It can help drivers in tracking children when they get off or get on the school bus.

The key technique in the system is real-time face recognition. It is an important popular field in computer vision. It has yielded many applications which include tracking school attendance, protecting schools from threats, diagnose diseases, unlocking phones, etc. It includes image pre-processing, real-time face detection, face alignment, feature extraction, and classification.

1.2 Application

The system can be installed on a portal of a school bus. By capturing people walking through the portal, it can recognize them and then store their names. The driver can obtain information about people by the system. Moreover, it can recognize strangers when they want to intrude on the bus. By monitoring people on the bus, it provides safety tips for the driver.

If a bus has two portals, one is used for entering and the other one is used for exiting. In that case, two cameras are needed to connect with the system. However, the theory is the same as the system with one camera. The number of people on the bus is equal to the number of people entering the bus minus the number of people exiting the bus.

The feature extraction algorithm, ResNet, can be implemented in some other mobile devices, such as mobile phones. Because the ResNet has fewer parameters, it is suitable for devices with limited memory.

1.3 Challenges

Challenges include the design of ResNet, real-time processing videos and the detection of strangers. The number of layers and the dimension of an embedding layer are very important in designing a ResNet. The memory size of our SD card is 16 GB which is very small when real-time processing videos. Therefore, the system needs more time to recognize faces.

How to detect a stranger is also a challenge when we design the system. A threshold value needs a lot of time to select. The other challenges contain:

- Data cleaning of the training dataset for ResNet;
- Crop out face are;
- Obtain multiple faces in a frame;
- Align faces using facial landmarks;

- Select classification algorithms;
- Configure the distance sensor and motion sensor.

1.4 Contributions

The system can launch the function of face recognition according to the distance between people and whether people are moving. A threshold distance can be set in the system. If people are in the range of the threshold distance, the system begins to recognize people. Otherwise, it does not work. This setting can save energy and reduce the work time of it. Thus, the system can intelligently select work time.

The ResNet, besides implemented in the system, can be used in some other mobile devices because it has a small number of parameters and it needs less time to extract facial features.

1.5 Organization

The rest of the thesis is organized as follows: Chapter II is the related work. It contains steps of face recognition, the present face detection algorithms, the theory of face alignment, algorithms of facial feature extraction. Algorithms of face classification. Chapter III describes system configuration and setup processes. It includes hardware setup and main python libraries used in the system. Chapter IV presents the methods and results of designing the system. It contains dataset selection and collection, image preprocessing, ResNet models of feature extraction, face classification on still images and face classification on real-time videos. Chapter V summarizes the thesis.

CHAPTER II:

RELATED WORK

2.1 Face recognition

Face recognition is a technology of identifying a person from a digital image or a video. Video is composed of a sequence of frames. Each frame is a digital image. Therefore, face recognition implemented on a video is based on face recognition on a digital image. Real-time face recognition is carried out by recording a video and identifying a person at the same time.

Face recognition is a hot research topic in recent years with the prosperous development of artificial intelligence. It is implemented by analyzing the appearance of people and utilizing pre-existing knowledge. The advantages of face recognition are no contact and uneasiness of perception. Compared to fingerprint recognition and iris recognition, it does not need fingerprint readers and iris scanning devices. More important, it does not need the cooperation of people and it is not easy for people to observe cameras.

However, face recognition has disadvantages, such as high computation and data storage. Additionally, people can change their appearance by making up and using accessories, then mislead face recognition. Nowadays, it is widely used in railway stations and hotels in China. It is also used in payment in China, but the use is not so widely. In America, it has many applications, for example, recognizing people on Facebook.

Steps of face recognition are, in most cases, face detection, face alignment, facial feature extraction, and face classification. Facial feature extraction is a key step in it. Several models currently available for facial feature extraction are based on neural networks [6]. This thesis tends to present a new model for extracting face features. Compared to previous models, the number of parameters is relatively small, and it is suitable to use in a family or school scenario. Its requirement for computing is not very

high. In the thesis, an intelligent in-vehicle safety and security monitoring system is designed based on Raspberry Pi, Pi camera, distance sensor, motion sensor, and screen. The system can recognize people when distance and motion satisfy the set condition. It monitors who gets on or off a vehicle and the number of people in the vehicle.

2.2 Face detection

Face detection is a prerequisite process of automatic facial image analysis including face recognition and verification, face tracking, face landmark detection, sentimental classification, etc. There are various methods to categorize algorithms of face detection. One method groups algorithm into four categories [7]: feature invariant approaches, template matching methods, knowledge-based methods, and appearance-based methods. Another method categorizes algorithms of face detection into two schemes [8]. Rigid templates and deformable models. The former one is to learn rigid templates that mainly on boosted cascades methods and deep neural networks. The latter one detects face via its parts.

Three popular face detectors are used to detect faces in current research and applications.

- Haar feature-based cascade detector in OpenCV;
- Local Binary Patterns (LBP) cascade detector in OpenCV;
- Deep Learning-based detector in Dlib.

2.2.1 Haar feature-based cascade detector

In 2001, the Haar feature-based cascade detector is presented by Paul Viola and Michael Jones in a paper, “Rapid Object Detection using a Boosted Cascade of Simple Features" [9][10]. It is used to determine whether there are any objects in images. In OpenCV, it is trained from many images which include negative images with no faces and

positive images with faces. As a face detector, it is a very influential approach in recent decades because it achieves high accuracy and processing images with high speed [11].

There are three key components of the detector.

- Integral images. It simplifies and accelerates the calculation on feature calculation.
- Adaboost algorithm. It selects important features from a large number of features to make the detector more efficient.
- Cascade of classifiers. It discards information of background and focuses on computation on promising face regions.

2.2.2 LBP detector

LBP face detector is also provided by OpenCV [12]. It is faster and higher detection rate than Haar cascade detector. Comparison between LBP detector and Haar cascade detector on single face images is shown in the Table 2.1 and Table 2.2.

Table 2.1: Result of Haar cascade detector [13]

Algorithm	Dataset	Detected Faces	Hit Rate	Detection Speed (ms)
Haar	Color FERET	976/1127	86.6%	235.4117
	MIT	1670/2000	83.5%	255.5048
	Taarlab	647/759	85.2%	231.5865
Overall		**3293/3886**	**84.7%**	**241**

Table 2.2: Result of LBP detector [13]

Algorithm	Dataset	Detected Faces	Hit Rate	Detection Speed (ms)
LBP	Color FERET	1004/1127	89%	95.44924
	MIT	1779/2000	89%	101.8864
	Taarlab	674/759	88.8%	104.9335
Overall		**3457/3886**	**89%**	**101**

2.2.3 Face detector in Dlib

Dlib is a C++ toolkit [14] that contains many machine learning algorithms to solve problems in a lot of domains containing robotics, mobile phones, embedded devices, etc. The face detector in Dlib uses a Maximum-Margin Object Detector [15] with convolutional neural networks (CNN). It is trained by a dataset that contains 7220 images and is from various datasets, for example, ImageNet, VGG, PASCAL VOC, Face Scrub and WIDER.

2.3 Face alignment

Face alignment is a form of data normalization. Before you train a machine learning model of face recognition, you may align faces in your dataset. In other words, you need to normalize a set of feature vectors via zero centering. These feature vectors are extracted from facial areas. Face alignment is a necessary step when Eigenface, Fisherface, LBP, and deep learning algorithms are used to recognize faces. There are two main steps [16] of it:

- Obtain the geometric structure of each face.
- Align faces by normalized rotation, scale, and translation from facial landmarks
- After face detection, every face is stored as an image which is represented by an array. The face is scaled that every face has approximately the same size. Additionally, face is rotated that the y-coordinates of eyes have the same value.

In the thesis, face alignment relies on a pre-trained facial landmark detector [17] in the Dlib library. The detector predicts 68 key points that map facial contour. Those key points are shown in Figure 2.1. However, we only use those12 points to align face images. 6 points represent the contour of the left eye and 6 points represent right eye. We get the coordinate of a center of an eye by coordinates of 6 points. Then we obtain the coordinate of a center of two eyes. The center of two eyes is a center of the rotation in a face image. By rotation, scaling, and translation, face images are normalized to implement face recognition [18]. Figure 2.2 shows that an original face with an angle of inclination is converted to an aligned face without inclination.

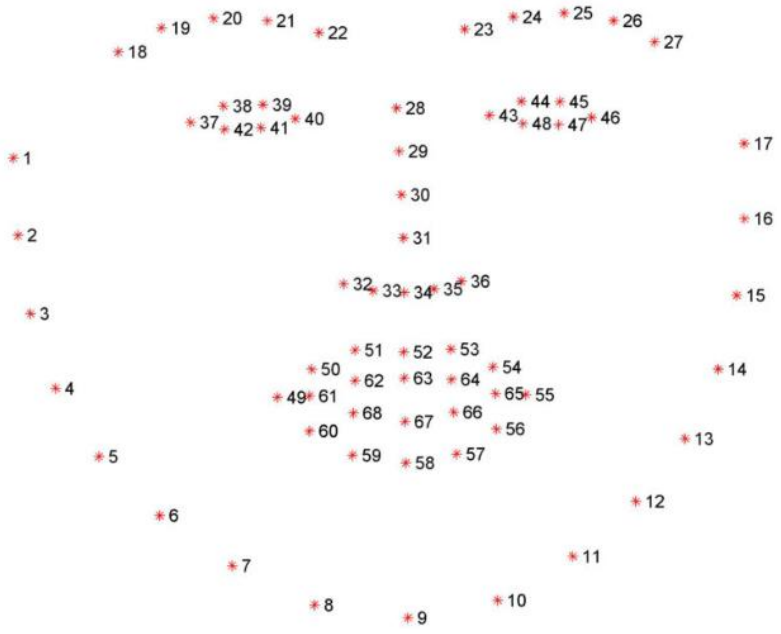

Figure 2.1: The 68 face land markers [19].

Figure 2.2: Face alignment: from an original image to an aligned face image.

2.4 Overview of facial feature extraction models

A face digital image is composed of many pixels. When processing a digital image, it is stored as an array, and a pixel is represented by a number in the array. A color image is represented by a three-dimensional array, and a gray image is represented by a two-dimensional array. In order to increase the speed of computing, a color image is often transformed into a gray image. Another reason is that color information does not help to detect edges or features [20]. The gradient of the intensity of luminance is significant to edge or pattern detection which is effective information to distinguish objects. Converting color images to gray images reserves gradient of intensity of luminance and avoid color information. Even though transformation from a color image to a gray image, important features still covered by unimportant information. Those important features are used to distinguish different faces. Thus, feature extraction is the key step of face recognition. Several models have been developed and are available now for the facial feature extraction.

2.4.1 Eigenfaces method

Eigenfaces used for recognition was developed by Sirovich and Kirby in 1987. The method is used for face classification by Mathew Turck and Alex Pentand in 1991. The method of eigenfaces is based on Principal components analysis (PCA). It projects face images onto a feature space in which each axis characterizes a significant variation among

known face images [21]. Important features are the eigenvectors of the set of faces. By projection operation, a face image is represented by a weighted sum of the eigenface features. When recognizing a face, it only needs to compare these weights to those of known face in a dataset.

PCA is a statistical technique to convert a set of data of possibly correlated variables into a set of data of linearly uncorrelated variables by the orthogonal transformation. The goal of it is feature dimensionality reduction. These linearly uncorrelated variables are called principal components. The first principal component has the biggest variance, and the second principal component has the second biggest variance, so on and so forth. Each succeeding principal component is orthogonal to the preceding principal components. q-dimensional subspace is created when q principal components are selected. However, principal components with small variances are not selected. Projections of original data on those principal components are considered uninformative information. We want to project original vector or data on to the q-dimensional subspace. The projections on the q-dimensional subspace are considered crucial information. An example of principal components of two-dimensional is shown in Figure 2.3.

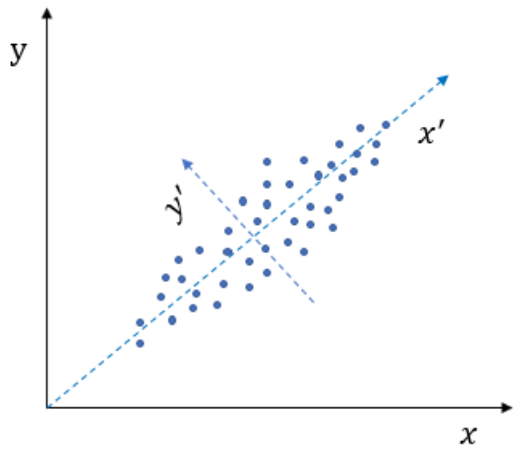

Figure 2.3: The principal components of two-dimensional data demonstrated by the blue and green lines[21]

Eigenfaces are the name of eigenvectors which are also called principal components. They are calculated by covariance matrix. Let a training set of face images be $T_1, T_2, T_3, \ldots T_n$. The average face is $\psi = \frac{1}{n}\sum_{k=1}^{n} T_i$. The difference between each image and the average image is $\Phi_i = T_i - \psi$. Let $M = [\Phi_1, \Phi_2, \ldots \Phi_n]$, then the covariance matrix is $F = MM^T$. Consider the covariance matrix is $F' = MM^T$ instead F. The eigenvector v_i can be calculated by $F'v_i = \mu_i v_i$. μ_i is the eigenvalue. The eigenvector u_i of F is

$$u_i = \mathrm{M}v_i \tag{2.1}$$

We want to reduce the dimension of each face image. If the dimension of each face image is N, now it is reduced to N'. By selecting the N' eigenvectors that have the largest associate eigenvalues. The N' eigenvectors make the N'-dimensional subspace. A new face image is projected into the N'-dimensional subspace, then we can get $E = [\alpha_1, \alpha_2, \ldots \alpha_{N'}]$. Now the new face is represented by the E which is used in face classification.

2.4.2 Method based on deep convolutional network

The history of image recognition using computers has more than fifty years. During the 1950s and 1960s, learning machines [22]brings a great achievement in machine learning [23]. A basic unit, called perception can converge to a value by finite iterations in a training process. However, these learning machines are not enough for complex tasks. Researchers attempt to increase the power of perceptions by stacking more layers. In 1986, backpropagation is presented. It is a method of training neural networks [24]. In 1989, backpropagation is first implemented in deep convolution neural networks (CNN). In 2012, AlexNet wins ImageNet Large Scale Visual Recognition Competition (ILSVRC) which is an image classification competition. Alex Krizhevsky designs the AlexNet that is CNN

[25]. CNN is proved to be very effective in image recognition. Then CNN is widely used in different image recognition [26]. A CNN architecture is shown in Figure 2.4.

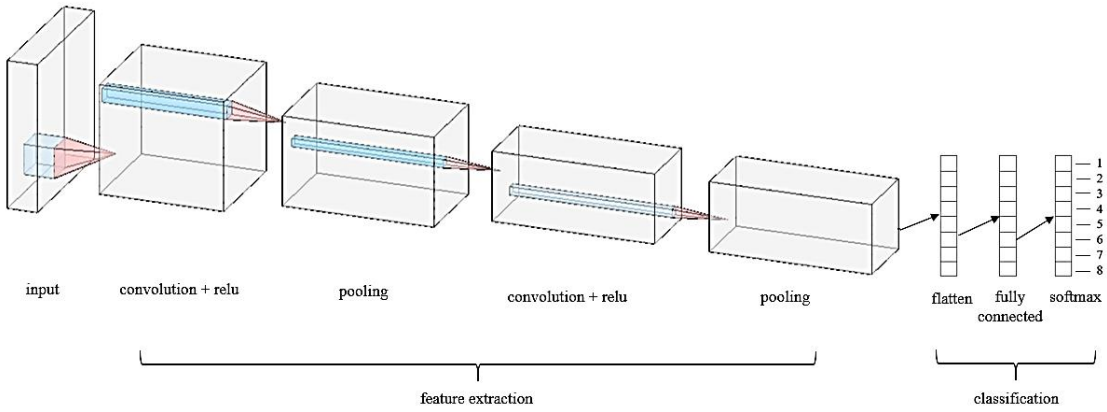

Figure 2.4: A CNN architecture that is used by number classification

Input is a digital image that is composed of pixels. If the image is a color image, the number of layers is 3 because each pixel has 3 color channels for red, green and blue. If the image is a gray one, the number of the layer is 1 as it has a single channel.

Convolution layer is composed of filters which are also called kernels or neurons. Each filter must have the same number of channels with the input image. For example, if the input image is a color image, each filter should have 3 channels. Each filter is connected a local region of the input image. The local region is called the reception field whose size is equivalent to the size of the filter. Each filter convolves each reception filed in the image and produces a 2-dimensional activation map if the input is a gray image. In fact, the calculation of convolution is dot product between filters and reception fields. An example of dot product between an image and a filter is shown in Figure 2.5. Then an activation function is implemented on the output of convolution for the reason those non-linear properties are added to CNN. Activation functions introduce non-linear transformation to

make CNN capable to learn complex projects. Without them, a convolution layer is same as a linear regression model. Three popular activation functions are used in practice. They are rectified linear units (ReLU), sigmoid function and tanh function.

ReLU function:

$$f(x) = \max(0, x) \tag{2.2}$$

Sigmoid function:

$$f(x) = \frac{1}{1+ e^{-x}} \tag{2.3}$$

Tanh function:

$$f(x) = \tanh(x) \tag{2.4}$$

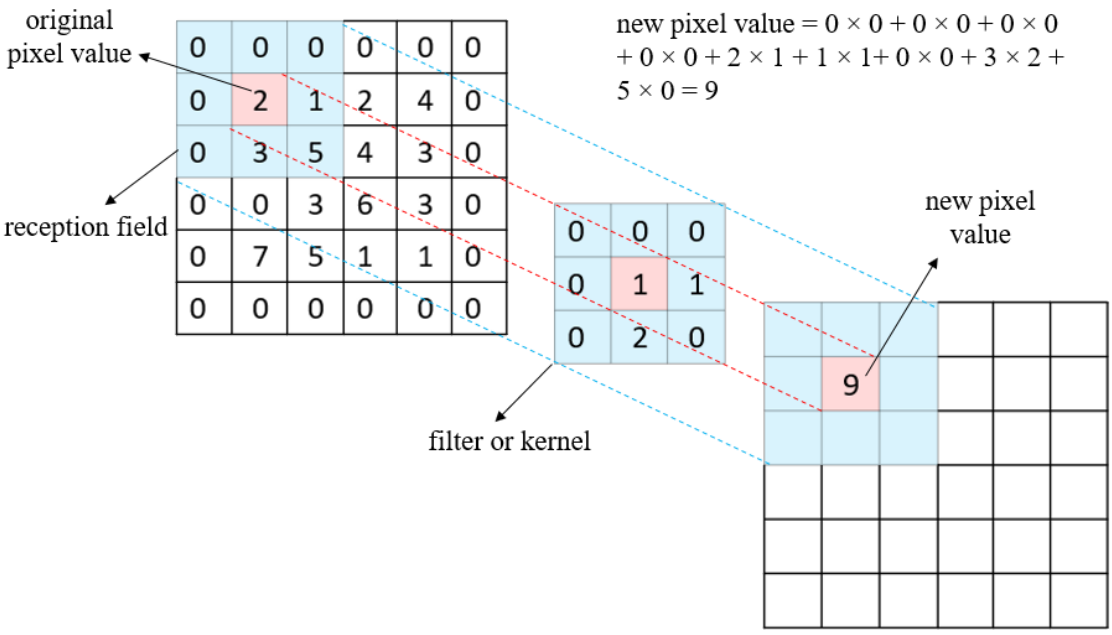

Figure 2.5: An example of the dot product between an image and a filter.

After the calculation of the convolution and activation function, an original image is transformed into new features that form a feature map. The feature map is used as input to the next layer on CNN.

The pooling layer is often inserted in between successive convolution layers. The purpose of pooling layers is to reduce the number of parameters and enhance the speed of computing. Max-pooling layers and average-pooling layers are usually used on CNN. An example of max pooling and average pooling is shown in Figure 2.6. People prefer max-pooling layer over average-pooling layer because a max-pooling layer overperforms an average-pooling layer in pattern recognition [27].

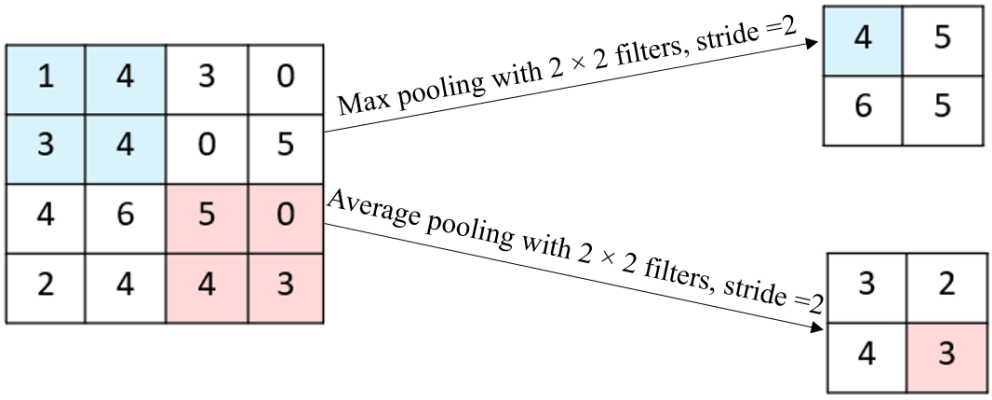

Figure 2.6: An example demonstrating max pooling and average pooling

The flatten layer is used to flatten the output of the final pooling layer or convolution layer. The output of the final pooling layer or convolution layer is, in most cases, a multi-dimensional array and called feature maps. A flatten layer transforms the multi-dimensional array into one-dimensional vector which is inputted to a full-connected layer [28][29]. An example of flatten layer is shown in Figure 2.7. The flatten layer in just a tool of reshaping data so that the data is compatible with the subsequent layer. Thus, it is in front of a full-connected layer.

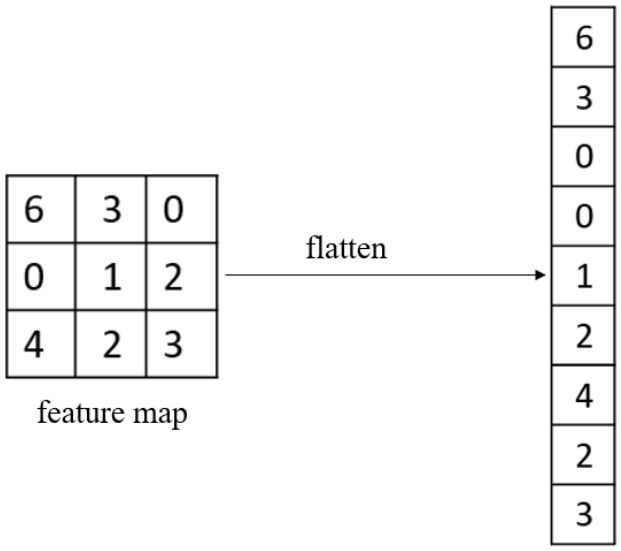

Figure 2.7: An example demonstrating the transformation of flattening

A full-connected layer is also called a dense layer. It takes the output of the previous layer and applies weights to predict the correct label. In general, non-linear activation function is implemented in the layer. The full-connected layers act as a classifier and assign probabilities for the input image in every category or label.

Most methods of face recognition are based on CNN. By changing architectures, convergence and loss function, different neural networks have been created in recent years. A method using triplet loss is presented by Florian Schroff el al. The deep convolutional network is Inception network. The network is trained according to face similarity. Faces of different persons have large distance and faces of the same people has small distances. The method first presents triplet loss in face recognition. The triple loss is shown in Figure 2.8. The approach has much greater representational efficiency [6]. It only uses a compact 128-dimensional vector to represent a face. In the triplet loss, an image x_i^a (anchor) represents a specific people. x_i^p (positive) represents all the other images with the same people. x_i^n

(negative)represents all the other images with different people. There the triple loss function is $||x_i^a - x_i^p||_2^2 + \alpha < ||x_i^a - x_i^n||_2^2$, $\forall(x_i^a, x_i^p, x_i^n) \in$ T [29]

α – a margin that is enforced between positive and negative pairs

T – the set of all possible triplets

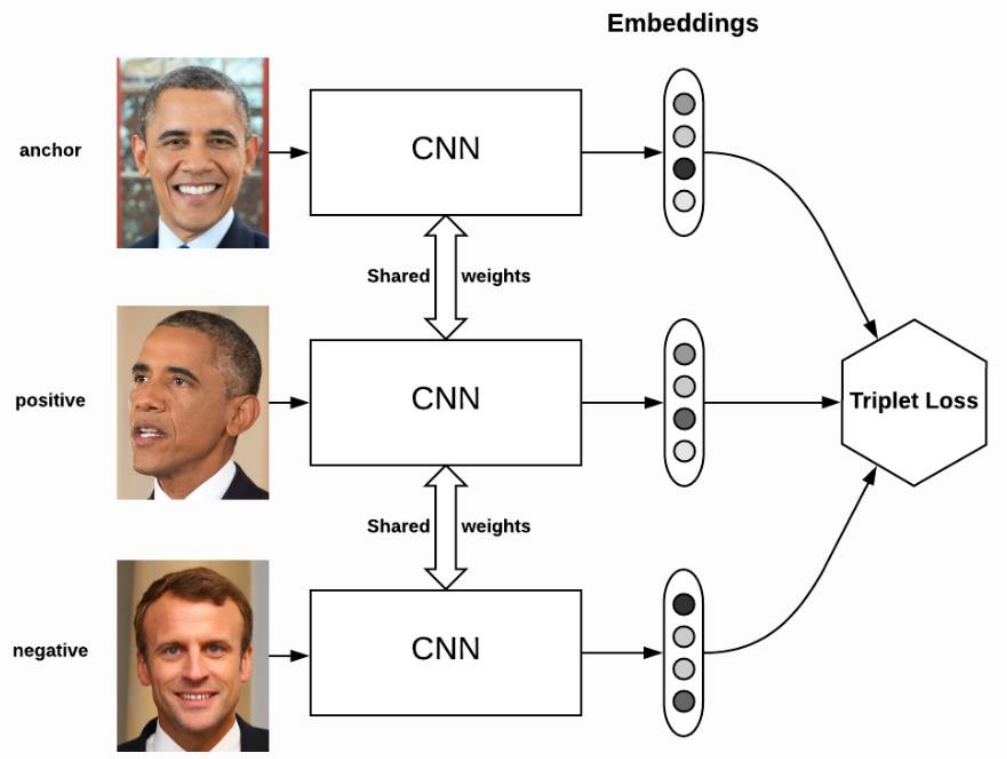

Figure 2.8: Triplet loss on anchor, positive and negative faces [30]

Six architectures, NN1, NN2, NN3, NN4, NNS1, and NNS2, are presented by Florian Schroff el al. NNS1 has 26M parameters, and NNS2 has 4.3M parameters. They are suitable for mobile devices. The method achieves good performance on LFW and YTF. However, triplet loss is difficult to be implemented in training neural networks.

Methods based on neural networks using a softmax cross-entropy loss makes training significantly easier and faster [31].

In Computer Vision and Pattern Recognition (CVPR) 2014, Yi Sun et al. presented DeepID (Deep hidden Identity features) for face verification. The DeepID can challenge tasks of multi-class face identification. Face features are extracted from the last hidden layer neuron activation of a CNN. CNN has four convolution layers, three max-pooling layers. After the last convolution layer, there is a layer that represents deep hidden identity features (DeepID). The last layer is a classical soft-max layer. Every input layer or image is 1 face patch. There are 60 face patches with ten regions, three scales for a face. Therefore, 60 convolution networks are needed to be trained. Each convolution network produces a 160-dimensional DeepID vector. The vector is horizontally flipped, then a new vector is obtained except that patches around the two mouth corners and two eyes centers. Those patches are processed by other methods. Thus, by training a convolution network on one patch of a face, a vector and its flipped counterpart are obtained. After 60 patches are trained, 120 DeepID vectors are gotten. Every vector has 160 dimensions. Those vectors make up a total DeepID vector that is composed of 19,200 values. The total DeepID does not use only a detected face, it uses different parts or patches of a face with three scales, then synthesizes all features extracted from those patches. In 2005, they presented two very deep neural networks for face recognition which are based on VGG and GoogleNet [32]. Accuracies are similar to the DeepID. The architecture of convolutional network for learning DeepID is shown in Figure 2.9. Ten face regions of medium scales used in DeepID are shown in Figure 2.10.

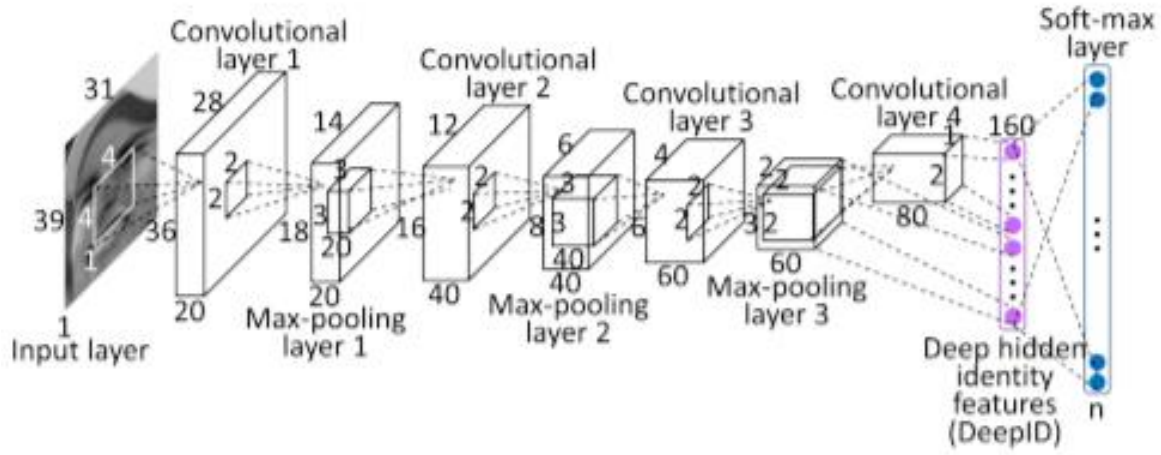

Figure 2.9: The architecture of convolutional network for learning DeepID [33]

Figure 2.10: Ten face regions of medium scales [33]

2.4.3 Method based on deep residual network (ResNet)

Kaiming He et al. presented a deep residual network that won first place in ILSVRC (ImageNet Large Scale Visual Recognition Competition) and COCO (Common Objects in Context) 2015 competition in all five main tracks. ResNet can largely overcome degradation in deep learning and Ease training in neural networks [34]. It has big potential in face recognition and overperforms traditional CNN [35]. A segment of the ResNet architecture for face recognition presented by Kaiming He is shown in Fig. 11. Block is an important feature in ResNet. It is implemented by "shortcut". Let an input value is x. The

desired mapping is h(x). If let nonlinear layers fit a mapping of f(x) = h(x) − x, then the mapping becomes x + f(x). The new mapping x + f(x) is realized by "shortcut". The demonstration of one block in ResNet is shown in Figure 2.12.

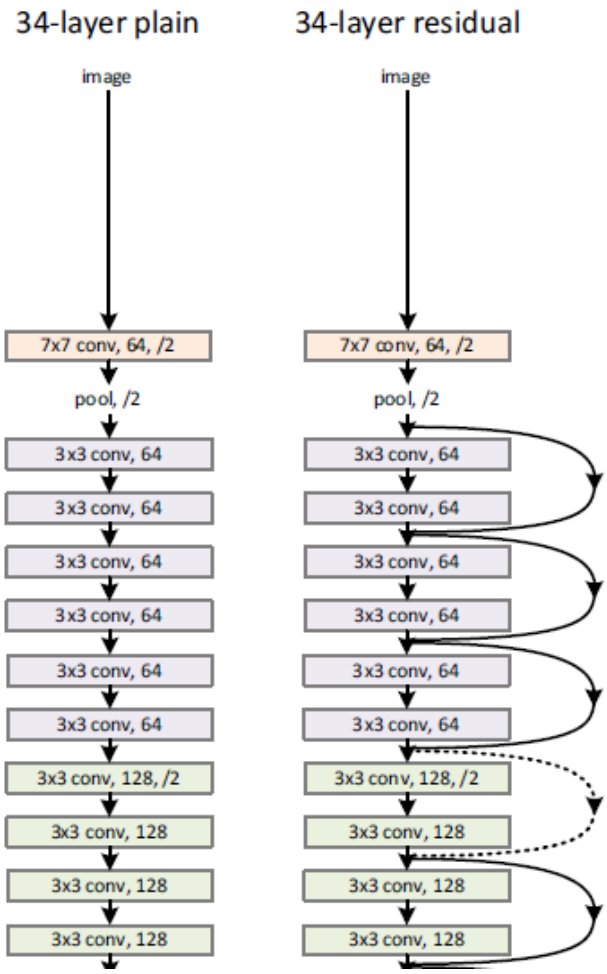

Figure 2.11: A segment of architectures for a ResNet (ImageNet) [34]

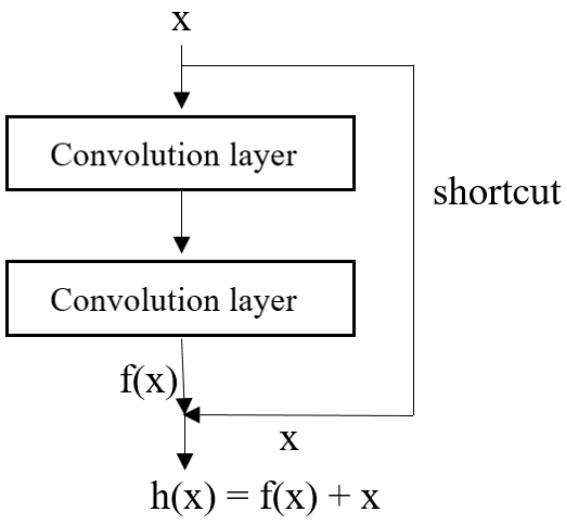

Figure 2.12: Explanation of a block in a ResNet[34]

2.5 Face classification

After feature extraction, face embedding is obtained. There are several classifiers that can be implemented in face classification. In the thesis, three classifiers, k-nearest neighbors' classifier (kNN), Support-vector machine classifier (SVM) and softmax classifier, are introduced. The three classifiers are supervised learning techniques.

kNN is a model that classifies a data point based on the k nearest neighbors around the data point. In 1951, Fix and Hodges presented a non-parametric method for classification in an unpublished US Air Force School of Aviation Medicine report. The method has been known as kNN. How kNN works is shown in Figure 2.13. It assumes that similar data points exist in proximity. It is similar to the proverb that birds of a feather flock together. For example, five neighbors are considered when k is equal to 5. Those neighbors are closest to the data point. Then the five neighbors or data points vote for the data point. The data point is classified by a majority vote of its neighbors. Neighbors are defined

according to the distance between two data points. Three distance functions are often used for calculation of distance. Let a point $X = (x_1, x_2, \ldots x_n)$ and Y $= (y_1, y_2, \ldots y_n)$.

Manhattan distance:

$$d(X,Y) = \sum_{i=1}^{n} |x_i - y_i| \tag{2.5}$$

Euclidean distance:

$$d(X,Y) = \sqrt[2]{\sum_{i=1}^{n}(x_i - y_i)^2} \tag{2.6}$$

Makowski distance:

$$d(X,Y) = \left(\sum_{i=1}^{n} |x_i - y_i|^p\right)^{1/p} \tag{2.7}$$

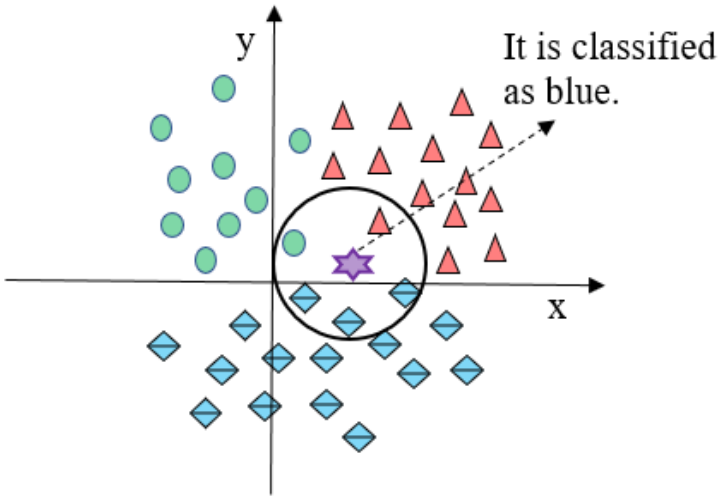

Figure 2.13: How data point is classified using kNN algorithm

The algorithm is easy to implement. It is effective when training data is large. It is robust to noisy data. However, it is needed to define the value of k which is the number of nearest neighbors. It needs to compute the distance of every data point in the training dataset to the data point which is classified [36].

SVM creates boundaries to divide data points in a training dataset into groups based on their labels [37]. SVM separates the training data into different categories by locating a

decision function. The decision function is selected according to its distance to the nearest data point in each category. It maximizes its distance to the nearest data point in each category. When no decision function can linearly divide the training dataset, a kernel transformation function is needed to project the training dataset on different dimensional spaces in order that the dataset can be linearly classified using a decision function. A slack error variable is used to create a decision function in case the dataset cannot be fully separated by kernel transformation [38]. How an SVM works is shown in Figure 2.14.

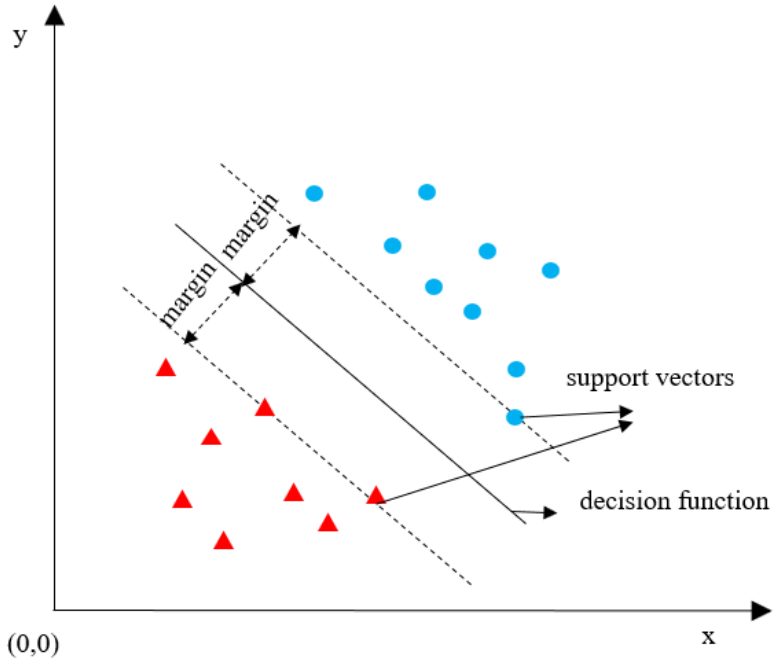

Figure 2.14: A support vector machine decision function (solid line) and 3 support vectors[38]

Softmax classifier is the generalization of binary logistic regression classifier. It is often used in multiple classifications and uses cross-entropy loss function [39]. If you classify a data point, it gives you probabilities for every class label. Let a mapping function is $f(x) = Wx$. A data point in a dataset is represented by x. It is input. W is a weight matrix.

The function f(x) is used to map a data point to its label. In general, the mapping function should be is f(x) = Wx + b, where b is the bias vector. However, in the thesis, f(x) = Wx is used to simplify the deduction.

$$z = \mathrm{f}(x) = \mathrm{W}x \tag{2.8}$$

Activation function:

$$t_i = e^{z_i} \tag{2.9}$$

The element at i-th position of the ground-truth vector

$$\widehat{y_i} = \frac{t_i}{\sum_{i=1}^{n} t_i} \tag{2.10}$$

cross-entropy loss function:

$$\mathrm{H}(\mathrm{y}, \hat{y}) = -\sum_{i=1}^{n} y_i \log \widehat{y_i} \tag{2.11}$$

z — the output vector of a mapping function

i — the i-th position, the maximum value is the number of categories

y — the ground-truth vector

y_i — the element at i-th position of the ground-truth vector

$\hat{y}$ — the predicted probability vector

$\widehat{y_i}$ – the element at i-th position of the predicted probability vector

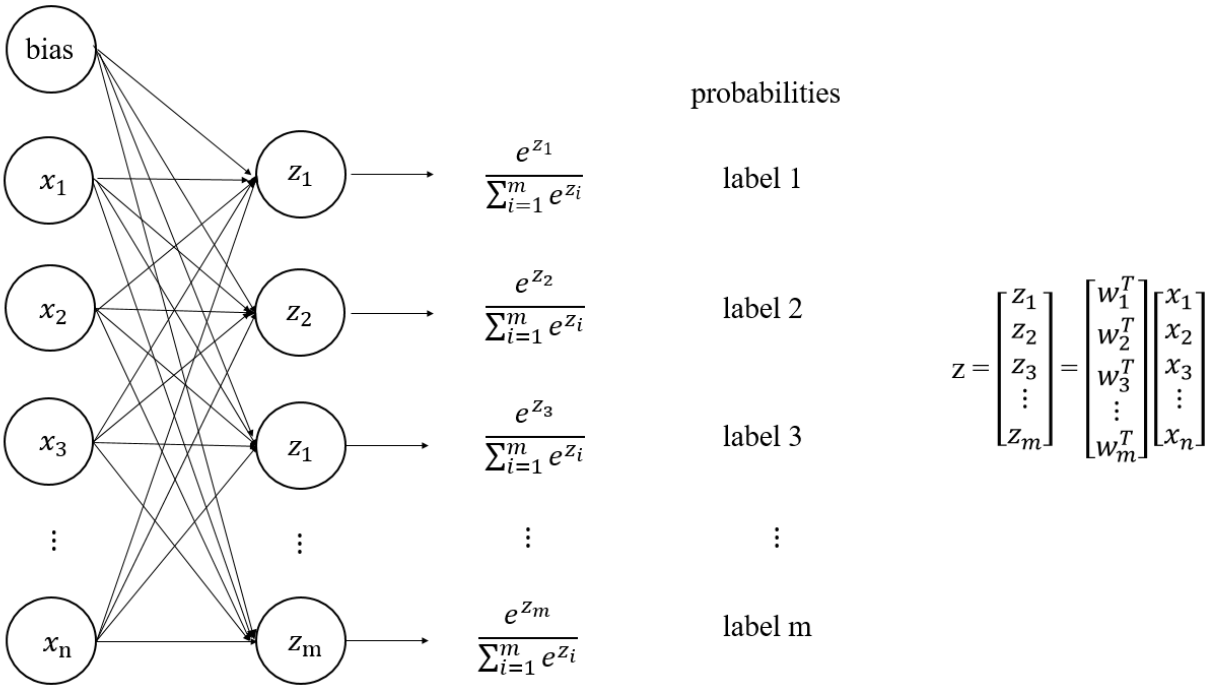

Figure 2.15: Multi-class classification using softmax classifier [40]

The softmax classifier is trained to minimize the loss function. When a data point is inputted to the classifier, we can obtain probabilities for each class. Multi-class classification using softmax classifier is shown in Figure 2.15.

CHAPTER III:

SYSTEM SETUP

The system contains hardware and software. The hardware consists of Raspberry Pi, camera module, wireless adapter, touch screen, distance sensor, and motion sensor. Raspberry Pi provides computing, like a mini-computer. The camera module collects images and videos. The wireless adapter helps the Raspberry Pi to connect Internet without a physical connection. The touch screen makes the system more portable. The distance sensor is used to measure the distance between an object and the Raspberry Pi. The motion sensor detects whether an object is in motion state. Those two sensors make the system have more functions. Whether face recognition is executed depending on distance and motion. Software is installed in Raspberry Pi through the interface of command line. The software consists of TensorFlow, Keras, and scikit-learn, OpenCV, Dlib, NumPy, and imutils.

3.1 Hardware

Raspberry Pi 3 Model B+, distance sensor, motion sensor, camera module, Wireless adapter, touch screen, HDMI cable, Ethernet cable, MicroSD card, and power adapter are used in the project. Raspberry Pi 3 Model B+ uses a Linux operating system and provides computing to the project. Camera module offers to capture real-time video. The wireless adapter is used to connect Raspberry Pi to the wireless network.

3.1.1 Raspberry Pi 3 Model B+

Figure 3.1: A Raspberry Pi 3 Model B+

Raspberry Pi 3 Model B+ shown in Figure 3.1 has 1.4GHz 64-bit quad-core ARM Cortex processor, CSI camera port for connecting a Raspberry Pi camera and Micro SD port for loading the operating system and storing data. A Raspberry Pi is shown in Fig.16. It supports dual-band 802.11ac wireless LAN, Bluetooth 4.2/BLE, 3×faster Ethernet, and Power-over-Ethernet support (with separate PoE HAT). It also provides remote access [47].

Specification of Raspberry Pi 3 Model B+ is as follows:

- 2.4GHz and 5GHz IEEE 802.11.b/g/n/ac wireless LAN, Bluetooth 4.2, BLE
- Broadcom BCM2837B0, Cortex-A53 (ARMv8) 64-bit SoC @ 1.4GHz
- Gigabit Ethernet over USB 2.0 (maximum throughput of 300 Mbps)
- Full-size HDMI
- Power-over-Ethernet (PoE) support (requires separate PoE HAT)
- Micro SD port for loading your operating system and storing data
- DSI display port for connecting a Raspberry Pi touchscreen display
- 4-pole stereo output and composite video port

- CSI camera port for connecting a Raspberry Pi camera
- 4 USB 2.0 ports
- 5V/2.5A DC power input
- The extended 40-pin GPIO header

3.1.2 Camera Module and Wireless adapter

Camera module supports Raspberry Pi Model A or B, B+, Raspberry Pi 3 and Raspberry Pi 3 Model B+. The angle of View is 54×41 degrees. Max frame rate is 30 frames per second. The maximum video resolution is 1080p and still picture resolution is 2592×1944. It has an Omnivision OV5647 sensor in a fixed-focus lens and an infrared cut-off filter. The Pi camera is connected with Raspberry Pi by a CSI port. A cameral model is shown in Figure 3.2.

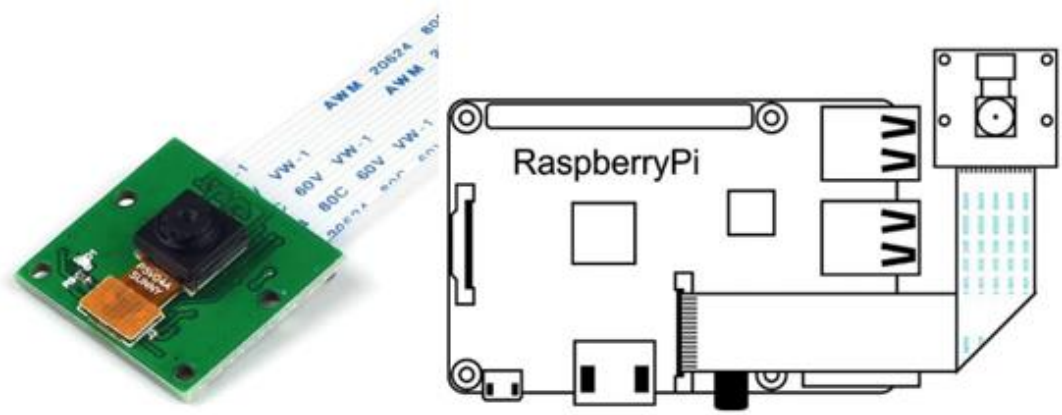

Figure 3.2: Camera module

A wireless adapter is a hardware device that is attached to a computer or other device to make it connect to a wireless network. It is shown in Figure 3.3. In the project, the wireless adapter is plugged into one of the USB ports on a Raspberry Pi to access the wireless network.

Figure 3.3: Wireless adapter

3.1.3 Ultrasonic Distance Sensor

A distance sensor has four GPIO (general-purpose input/output) pins which are connected to GPIO pins of Raspberry Pi. The four pins are VCC (input power, 5V), TRIG (trigger input), ECHO (echo output) and GND (ground). There are 40 GPIO pins in a Raspberry Pi which is called GPIO header. VCC and TRIG are directly connected to Pin 2 and Pin 12 of the GPIO header respectively. A signal from the distance sensor should be converted from 5V to 3.3V. Therefore, two resistors are needed. The ECHO is connected to one end of a 680 Ω resistor, then the other end of the resistor is connected to Pin 18 by a jumper wire, at the same time, connected to one end of a 1000 Ω resistor. The other end of the 1000 Ω resistor is connected to the GND. Then the GND is connected to Pin 39. The connection between the distance sensor and the Raspberry Pi of our system is shown in Figure 3.4.

Table 3.1: Specification of a distance sensor

model	HC-SR04
working voltage	5V DC
resolution	0.3 cm
frequency	40 kHz
range	2cm – 300cm
dimension	45 mm ×20 mm ×15mm

Table 3.1 shows the specification of a distance sensor. Distance calculation is by signal emission. The function of calculating distance is given as the follows,

$$\text{speed} = \frac{distance}{time/2} \tag{3.1}$$

where speed is 343 m/s and time is an interval for the signal from emission to reception.

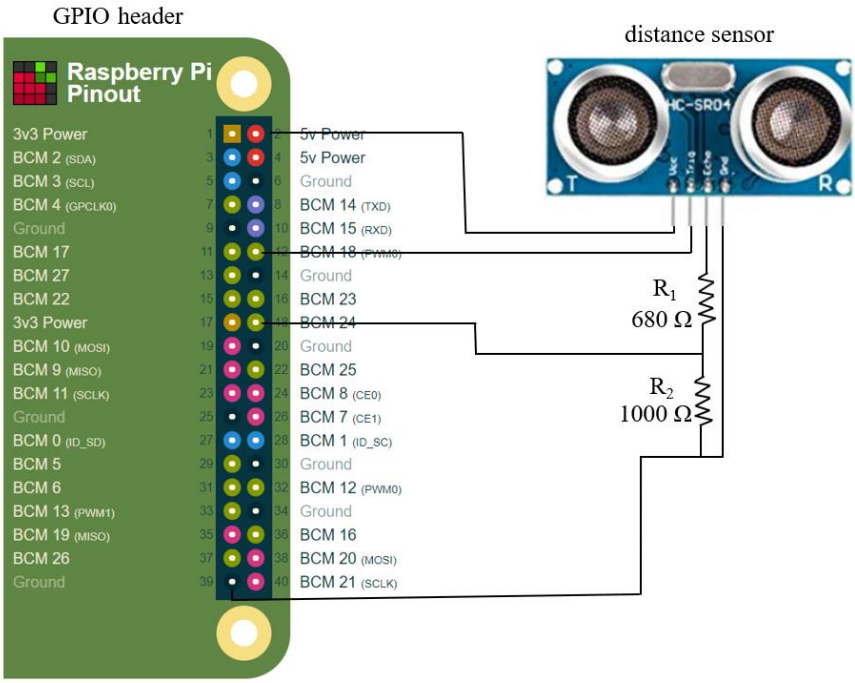

Figure 3.4: Connection of a distance sensor and Raspberry Pi

3.1.4 Motion Sensor

The motion sensor, in the thesis, is a PIR (passive infrared) motion sensor that can detect movement of objects. Objects emit heat in the form of infrared rays. The level of infrared ray changes when an object is in motion. The sensor can detect changes of infrared radiation from objects and then detect movement. It does not provide information about who or what is in movement. The specification of the motion sensor is shown in Table 3.2. The sensor has three pins which are VCC (input power), OUT (output) and GND (ground).The connection between the motion sensor and the Raspberry Pi of our system is shown in Figure 3.5.

Table 3.2: Specification of a motion sensor

model	HC-SR501
working voltage	4.5 – 20V DC
delay time	0.3 – 200s
board dimension	32 mm × 24 mm

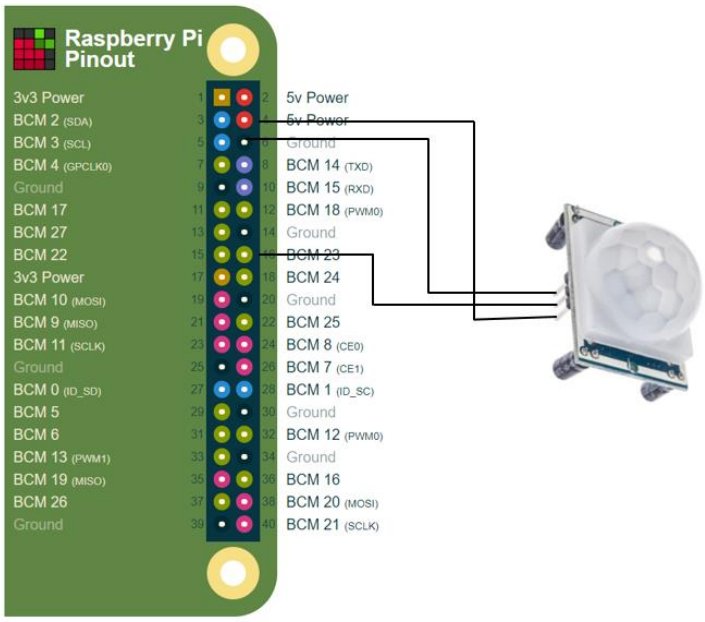

Figure 3.5: Connection of a motion sensor and Raspberry Pi

3.2 Hardware Setup

As a single-board computer, Raspberry Pi needs to be installed an operating system and libraries. Raspbian is installed in the system. It is an official supported operating system for all models of the Raspberry Pi. We use Python 3.5.6 as our programming language. Steps of setup are as follows,

Step 1. Insert an SD card into the Raspberry Pi, then install Raspbian. Downloaded Raspbian from the website, https://www.raspberrypi.org/downloads/.

Step 2. Create a Python virtual environment and install NumPy, imutilsTensorFlow and Keras.

Step 3. Download OpenCV and opencv_contrib libraries, install these libraries in the Raspberry Pi.

Step 4. Install the library dlib. A facial landmark detector is used in the system.

Step 5. Open the Raspberry Pi configuration and make that the camera is enabled. Test the cameral module by the command line *raspistill -o Desktop/image.jpg*.

Step 6. Install RPi.GPIO library to read and write pins on the GPIO header.

3.3 Software

3.3.1 libraries for deep learning and machine learning

The libraries of TensorFlow, Keras, and scikit-learn are used to construct neural networks and classification models.

TensorFlow is a powerful framework for machine learning and deep learning. It was released by the Google Brain on November 9, 2015, with the Apache 2.0 open source license. People can free use, change and distribute modifications. The framework supports C++, Python and et al. In the system, we use Python. TensorFlow supports GPUs and CPUs. It, as the name indicates, manipulates tensors. A tensor is a vector and matrix which

has n-dimensions. TensorFlow codes contain two parts. The first part is building graph that represents the data flow of computations. The other part is executing a session that runs the graph. TensorFlow is a low-level mathematical framework. But Keras is a high-level framework with encapsulation. The simplicity of Keras comes from many intuitive functions [41]. It is built on top of TensorFlow, in other words, TensorFlow offers the backend for it. Compared TensorFlow, it is a high-level library. When people install Keras, TensorFlow is needed to be installed at first. In the thesis, we use Keras to construct neural networks because of friendly implementation and fast prototyping, easy extensibility and work with Python.

Scikit-learn is a simple and efficient tool for machine learning. It is built on NumPy, SciPy, and matplotlib. Additionally, it is an open-source Python library that contains many algorithms implemented in classification, clustering, regression, dimensionality reduction, model selection and preprocessing. The library was publicly released in February 2010.

3.3.2 OpenCV and Dlib

OpenCV (Open Source Computer Vision Library) is an open-source library that provides more than 2500 algorithms in the field of computer vision. It owns Python, C++, MatLab and Java interfaces. We use a pre-trained Haar-Cascade algorithm from OpenCV for face detection and a VideoCaputre object for reading frames of videos.

Dlib is a C++ toolkit that contains many machine learning algorithms to solve problems in a lot of domains containing robotics, mobile phones, embedded devices, etc. It is also free of charge because of its open source license. In the thesis, a pre-trained landmark's facial detector model is implemented for face alignment.

2.3.3 NumPy and imutils

NumPy is a fundamental package in Python. Functions of it include computation of multi-dimensional array and matrices, complex linear algebra, Fourier transformation

and so on. In the thesis, it used to represent images and process frames. For example, a gray image is represented by a 2-dimensional array.

Imutils is a library containing a set of efficient functions. These functions are related to image processing, for instance, rotation, affine transformation, resizing, translation, edge detection and so on. We synthesize those functions to achieve face alignment.

CHAPTER IV:

METHODS

In order to meet the function of face recognition of the system, we need a model for facial face extraction and another model for face classification. The work contains dataset selection and collection, image preprocessing, ResNet Design, ResNet training, and classification models training. The workflow of face recognition is shown in Figure 4.1. First, we select LFW dataset to train our ResNet model of feature extraction. Second, we optimize the model by changing the number of layers and dimension of the embedding layer. After we obtain the model, the last layer – dense layer is removed. Thus, when a face image is inputted to the model, the output is a 256-dimensional face embedding instead of a face label. A model is just a tool for dimension reduction. The function is similar to PCA. When we want to classify or identify a face, the tool is used for feature extraction. Then another dataset – Chokepoint is used for face classification. By the ResNet model, every data is converted to a face embedding. These face embeddings are input of a kNN model and softmax classification model. By training the two models, a predicted name is given after a face image is inputted to each of these two models. The function of face recognition makes the system know who is in the vehicle or not.

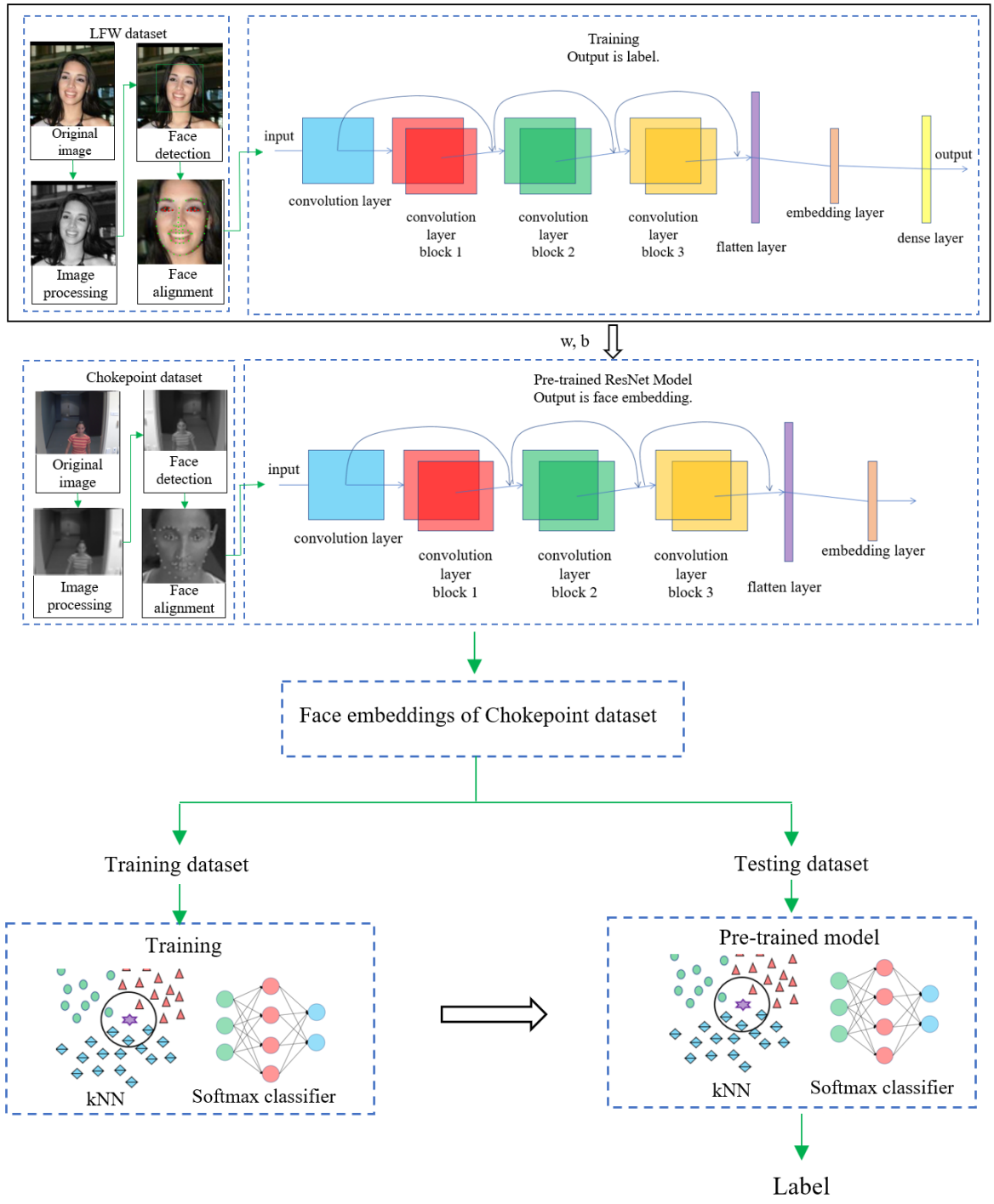

Figure 4.1: The workflow of face recognition

4.1 Dataset selection and collection

Three datasets are used in the thesis. They are the LFW dataset, Chokepoint dataset, and Delta dataset, respectively. The reasons why these three datasets are selected and collected are based on the specific function of the system. The LFW dataset is downloaded from the website of University of Massachusetts. It is used for training ResNet models which are tools of feature extraction. The Chokepoint dataset is used for face classification on images that simulate a real scene. The third dataset – Delta dataset is implemented for face classification on real-time videos. It also simulates a real scene. People entering a door is like people getting on a bus. Capturing videos and classifying faces are executed at the same time. The Delta dataset is created by us in a lab. Therefore, these three datasets have different uses.

4.1.1 Analysis of LFW dataset

Presently, no standard dataset has been built for training models of facial feature extraction. When we want to train a model, the selection of a reasonable dataset is the first thing we need to do. Reasonability means the dataset is suitable for your specific case. If your device or system has a very powerful ability of computation, more complex models or more data means higher accuracy. Our system uses the Raspberry Pi to real-time process video. Based on the fact, a very complex model may largely decrease processing speed or lead to crash. A large dataset, for example, several million images, is needed to complex models that may stack dozens of layers. Thus, a large dataset is not so practical.

In recent years, researchers have presented dozens of models. These models are trained on very large datasets. Table 4.1 shows the training datasets used for facial feature extraction. Each of datasets has from several million to dozens of million images. Several days are needed to train a model. Most of these models have big size, about 30 layers or

more, with millions of parameters. They are used to implement face recognition in LFW dataset. The LFW has 13233 images and 1680 people with two or more images [42]. The system, in the thesis, is used to monitor people on a school bus. In most cases, the number of people is usually not beyond 50 on a school bus. We do not need recognize so many people. Therefore, it is not needed to design models with many layers. These complex models may not adaptable to Raspberry Pi or some other mobile devices. Similarly, due to relatively simple models designed in the thesis, we do not need large datasets that have several millions of images. Additionally, the system, in the thesis, is used to monitor people on a school bus. In most cases, the number of people is usually not beyond 50 on a school bus. Therefore, we use LFW as the dataset for training models of ResNet.

Table 4.1: Training dataset in face feature extraction

Name	Model	Number of images	Number of people	Availability
CASIA-WebFace	SphereFace [43]	494,414	10,575	Public
Social Face Classification (SFC) dataset	DeepFace [44]	4.4 million [44]	4,030	Private (Facebook internal dataset)
VGGFace2	VGGFace [45]	3.3 million[46]	9,131	Public
Google dataset	FaceNet [6]	260 million [47]	8 million	Private
CelebFaces+	DeepID3	0.2 million	10,177	Public
Baidu dataset	CNN-9	1.2 million [48]	18,000	Private

The goal of the system is to monitor people when they get on or off a vehicle. After training the LFW, a model is gotten. The model can transform a face image into a face embedding. When the system real-time recognizes people, we need to store face embedding of those people in the system beforehand.

4.1.2 Analysis of Chokepoint dataset

To test our ResNet model, we need another dataset that can imitate people get on or off the vehicle. ChokePoint Dataset [49] is used in the thesis. It is a video dataset designed for experiments in people's identification in a real-world environment. There are three cameras that are placed in portals. One of the three cameras is possible to capture near-frontal face. When people are walking through a portal, his or her face is captured. Light, pose and sharpness may change in videos. Misalignment also exits in videos. There are two parts to the dataset. One part consists of 25 people (6 females and 19 males). Another part consists of 29 people (6 females and 23 males). In these videos, the frame rate is 30 fps and resolution is 800 ×600 pixels. One person appearing in a frame is shown in Figure 4.2. Multiple people appearing in a frame is shown in Figure 4.3.

Reasons [50] of selecting ChokePoint Dataset

- It consists of both one face in a frame and multiple faces in a frame.
- The number of people in the dataset is like the number of people on a school bus.
- People are walking through a portal is like that people enter a school bus.
- Variations of pose, sharpness, illumination are like real-world surveillance conditions.

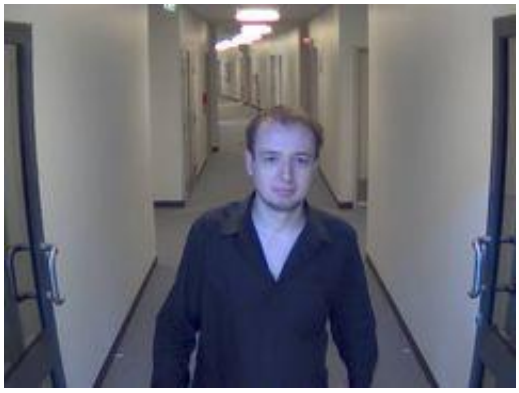

Figure 4.2: One people in a frame

Figure 4.3: Multiple people in a frame

4.1.3 Collection of Delta dataset

The role of the system is real-time monitoring people getting on or off vehicles. To simulate the real scene, we collect 120 images of 4 people. Every people have 30 images in which faces are at different angles. Therefore, the system can real-time recognize the 4 people when their faces are captured by Pi camera. All images of a specific person are stored in a folder named his or her name. The number of folders is equal to the number of people in the dataset. The storage of images is shown in Figure 4.4.

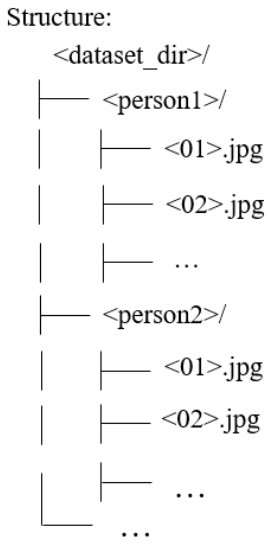

Figure 4.4: The storage format of images used in real-time face recognition

4.2 Image Preprocessing

In the thesis, image preprocessing contains color to grayscale conversion, face cropping, and affine transformation.

Color to grayscale conversion is implemented by a function in OpenCV. The function is gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY). Images or videos captured by cameras are color images. Each pixel in an image is described by three parameters. The three parameters represent the intensity of red, green and blue, respectively. Each parameter is an integer which is between 0 and 255. However, each pixel in a gray image is defined by a single parameter that describes the intensity of luminance or an amount of light. The parameter ranges from 0 to 255. 0 represents black at the weakest intensity and 255 represents white at the strongest intensity. Convert color images to gray images because the following reasons:

(a) Compared to color images, each pixel of a gray image just has one parameter, so a gray image has fewer dimensions. Gray images need less amount of computing and reduce the complexity of a model [51].

(b) Color information does not help to detect edges [20]. The gradient of the intensity of luminance is significant to edge or pattern detection which is effective information to distinguish objects. Converting color images to gray images reserves gradient of the intensity of luminance and avoid color information.

Face cropping is that facial regions are cropped from images and resized to a fixed size as the input of the next stage. Information on background, hair, bodies are not necessary for face recognition and not considered. Faces are only effective areas for recognition. These facial regions are detected by Haar feature-based cascade detector from OpenCV. The function is objects = cv2.CascadeClassifier(image). An example of face cropping is shown in Figure 4.5.

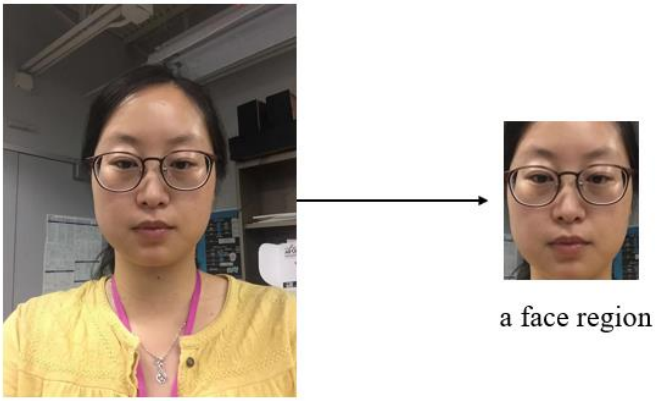

Figure 4.5: An example of face cropping in preprocessing images

Affine transformation is to align face. Steps of face alignment are as follows,

- Download the 68-facial landmark predictor from Dlib.
- Get coordinates of 6 points on the left eye and coordinates of 6 points on the right eye. The coordinates of 6 points on the left eye are $(x_1,\ y_1)$, $(x_2,\ y_2)$, $(x_3,\ y_3)$, $(x_4,\ y_4)$, $(x_5,\ y_5)$, $(x_6,\ y_6)$. The coordinates of 6 points on the right eye are $(x_7,\ y_7)$, $(x_8,\ y_8)$, $(x_9,\ y_9)$, $(x_{10},\ y_{10})$, $(x_{11},\ y_{11})$, $(x_{12},\ y_{12})$.

- Compute the center of each eye.

$$(x_l, y_l) = (\frac{x_1+x_2+x_3+x_4+x_5+x_6}{6}, \frac{y_1+y_2+y_3+y_4+y_5+y_6}{6}) \quad (4.1)$$

$$(x_r, y_r) = (\frac{x_7+x_8+x_9+x_{10}+x_{11}+x_{12}}{6}, \frac{y_7+y_8+y_9+y_{10}+y_{11}+y_{12}}{6}) \quad (4.2)$$

- Compute the angle between the two centers and the distance between the left eye and the right eye.

$$\alpha = \arctan\left(\frac{dy}{dx}\right) = \arctan(\frac{y_r - y_l}{x_r - x_l}) \quad (4.3)$$

$$\mathrm{d} = \sqrt[2]{(x_r - x_l)^2 + (y_r - y_l)^2} \quad (4.4)$$

- Calculate the location of the desired right eye based on the desired left eye location and the desired face width. These two arguments are given in default. The desired left eye location is not x-coordinate or y-coordinate but is a ratio. In the thesis, the setting value of it is (0.35, 0.35). It means the desired left eye is at 0.35 if the desired face width is 1.00. The setting value of the desired face width is 256. Thus, the location of the desired right eye is (0.65,0.65). The distance between the desired right eye and the desired left eye is as the following,

$$d_{desire} = 256 \times (0.65 - 0.35) = 76.80$$

- Calculate the scale for converting an original face width to the desired face width.

$$\mathrm{s} = \frac{d_{desire}}{d} \quad (4.5)$$

- Computing the center of the two eyes in the original face

$$\mathrm{c} = (\frac{x_r+x_l}{2}, \frac{y_r+y_l}{2}) \quad (4.6)$$

- Rotate the original face to make the connection line between the two eyes horizontal, then scale the face by using a function in OpenCV. The function is

  m = cv2.getRotationMatrix2D (c, α, s), m is the aligned face.

4.3 ResNet models of feature extraction

4.3.1 ResNet Design Process

The Design of ResNet contains what layers and how many layers are selected for a network. When an image is inputted to a network, in most cases, the first layer that it is fed to is the padding layer. The padding layer can make the design of a network easier because it can preserve the height and width of the input. In other words, it controls the shrinkage of dimension after using filters that are larger than 1×1. It also conserves information on the boundary. It is known that a convolution layer is used to extract edge pattern. Batch normalization is adopted in the architecture of ResNet for reasons that it induces more stable and faster training. Max-pooling layer is done to reduce dimension of data and mitigate over-fitting. In the thesis, an embedding also named bottleneck layer, is used to obtain a representation of an image by reducing nonlinear dimensionality. Embedding is inputted to a classification model instead of face images to achieve face recognition. Then a softmax classifier categorizes face images. The size of input image is 224 × 224 ×1 (height × width × channel) since gray image owns one channel. Our face recognition ResNet model contains the above layers.

The number of convolution layers of a ResNet, in general, from several to more than one hundred. A block is composed of several sequent convolution layers. In most cases, two or three layers compose a block. In the thesis, a block consists of two convolution layers. We compared performance of ResNet with two blocks, three blocks and four blocks, and select the ResNet with three blocks. The performance of the ResNet with different blocks is shown in Figure 4.6.

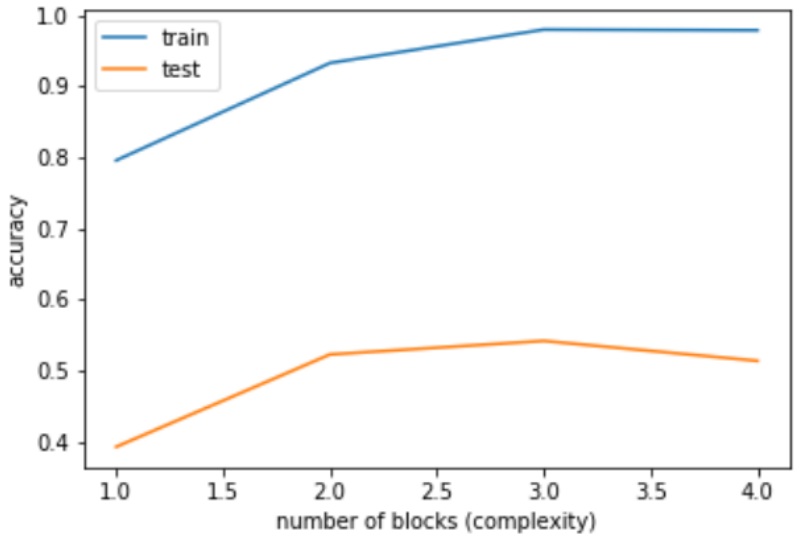

Figure 4.6: The accuracy of ResNet with different number of blocks

The ResNet with three blocks have a total 7 convolution layers, a flatten layer, a normalization layer, a full-connected layer. Full-connected layers are also called dense layers. The size of each input image is 224 × 224 pixels.

The dimension of a face embedding is also a significant consideration for ResNet. When we use 512-dimensional embedding and 256-dimensional embedding, the test accuracy is 71.50% and 72.06%, respectively. The 256-dimensional embedding is selected because it has less computation cost.

We trained the model using a subset of LFW. Each person has 7 or more images in the subset. Every people with less than 7 images in LFW is not trained. With increasing the number of images of each person, the accuracy of prediction increases very little when the number is bigger than 7. The prediction accuracy with different numbers of images for a person is shown in Figure 4.7. We use Adam optimizer with learning rate 0.001 and cross-entropy loss [31]. Kernel's initializer is set as “He normal”.

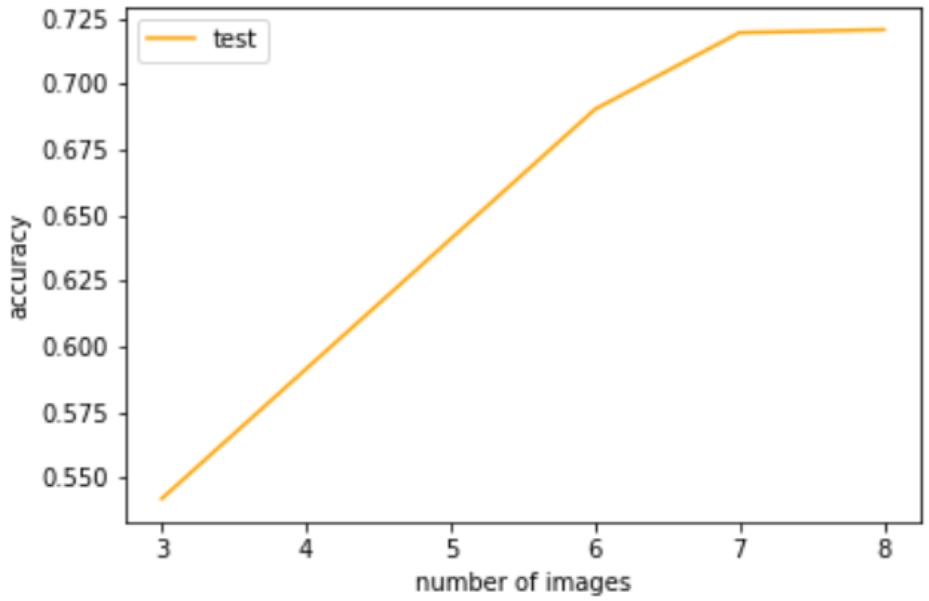

Figure 4.7: Testing accuracy with different number of images for one person

4.3.2 ResNet architecture

The architecture of the model is summarized by using the Kera function model.summary(). It includes the type, output dimensionality and the number of parameters of each layer which help people to design the size of filter, stride, size of the max-pooling windows and needed computational memory. Table 4.2 is shown the architecture of the ResNet model which is used as a tool for extracting facial features from images. The visualization of the architecture is shown in Figure 4.8.

Table 4.2: A summary representation of ResNet model

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	(None, 224, 224, 1)	0
zero_padding2d_1 (ZeroPadding2D)	(None, 230, 230, 1)	0
conv2d_1 (Conv2D)	(None, 112, 112, 32)	1600
batch_normalization_1 (BatchNormalization)	(None, 112, 112, 32)	128
max_pooling2d_1 (MaxPooling2D)	(None, 56, 56, 32)	0
conv2d_2 (Conv2D)	(None, 56, 56, 32)	9248
batch_normalization_2 (BatchNormalization)	(None, 56, 56, 32)	128
conv2d_3 (Conv2D)	(None, 56, 56, 32)	9248
batch_normalization_3 (BatchNormalization)	(None, 56, 56, 32)	128
add_1 (Add)	(None, 56, 56, 32)	0
conv2d_4 (Conv2D)	(None, 28, 28, 64)	18496
batch_normalization_4 (BatchNormalization)	(None, 28, 28, 64)	256
conv2d_5 (Conv2D)	(None, 28, 28, 64)	36928
conv2d_6 (Conv2D)	(None, 28, 28, 64)	18496
batch_normalization_5 (BatchNormalization)	(None, 28, 28, 64)	256
batch_normalization_6 (BatchNormalization)	(None, 28, 28, 64)	256
add_2 (Add)	(None, 28, 28, 64)	0
conv2d_7 (Conv2D)	(None, 14, 14, 128)	73856
batch_normalization_7 (BatchNormalization)	(None, 14, 14, 128)	512
Layer (type)	Output Shape	Param #

conv2d_8 (Conv2D)	(None, 14, 14, 128)	147584
conv2d_9 (Conv2D)	(None, 14, 14, 128)	73856
batch_normalization_8 (BatchNormalization)	(None, 14, 14, 128)	512
batch_normalization_9 (BatchNormalization)	(None, 14, 14, 128)	512
add_3 (Add)	(None, 14, 14, 128)	0
max_pooling2d_2 (MaxPooling2D)	(None, 2, 2, 128)	0
flatten_1 (Flatten)	(None, 512)	0
Bottleneck (Dense)	(None, 256)	131328
embedding (Lambda)	(None, 256)	0
dense_1 (Dense)	(None, 217)	55769

Total params: 579,097
Trainable params: 577,753
Non-trainable params: 1,344

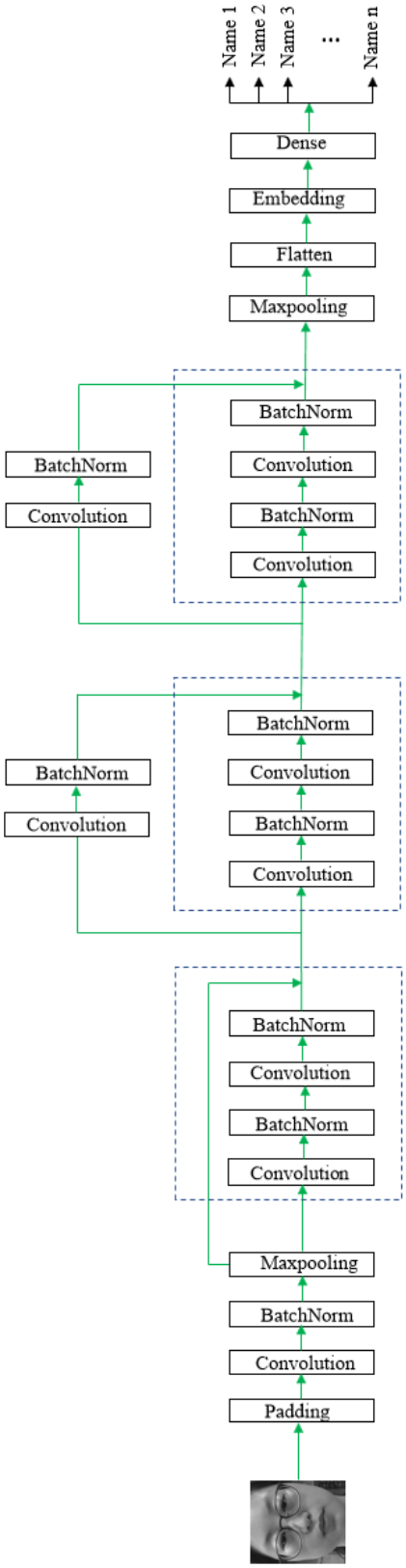

Figure 4.8: Architecture of the ResNet

4.3.3 ResNet training process

The dataset is LFW in which only people with 7 or more face images are selected. We divide the dataset into two parts – the training dataset and the testing dataset. We use a function to split the whole dataset because different people have different numbers of images. One person means one class here. The function is shown as the following,

$$d_{train} = [\, n \times (1 - \alpha)] \tag{4.7}$$

$$d_{test} = n - d_{train} \tag{4.8}$$

d_{train} – the number of training images of a people

d_{test} – the number of testing images of a people

n – the number of images of a people

α – the split ratio

In the thesis, the split ratio is 0.05. For instance, the number of training images of a people is 9 and the number of testing is 1, assuming the number of all images of the people is 10. The dataset used to train the ResNet model has a total of 4822 images belonging to 217 classes. The training dataset contains 4482 images and the testing dataset has 340 images.

This loss function is categorical cross-entropy also named softmax loss. It uses both softmax activation and a cross-entropy loss. The function is shown in the introduction of the thesis.

Mini-batch SGD (Stochastic gradient descent) is the gradient descent algorithm implemented to minimize loss function and update the weights of the ResNet. It is a trade-off between SGD and batch SGD. We need to set the two parameters – epoch and batch size for mini-batch SGD. The epoch is set as 160 which means the training dataset is passed forward and backward through the ResNet model 160 times. The batch size is 32. The API

ImageDataGenerator generates batches of tensor image data to feed the ResNet model. Batch size is the number of training images in a batch. The training dataset is divided into batches because the entire dataset cannot be passed to the ResNet at one with the limitation of CPU or GPU memory. There are 141 batches for one epoch in training ResNet. It means there are 141 iterations in one epoch. This history of training and testing accuracy and loss of the ResNet model are shown in the Figure 4.9 and Figure 4.10.

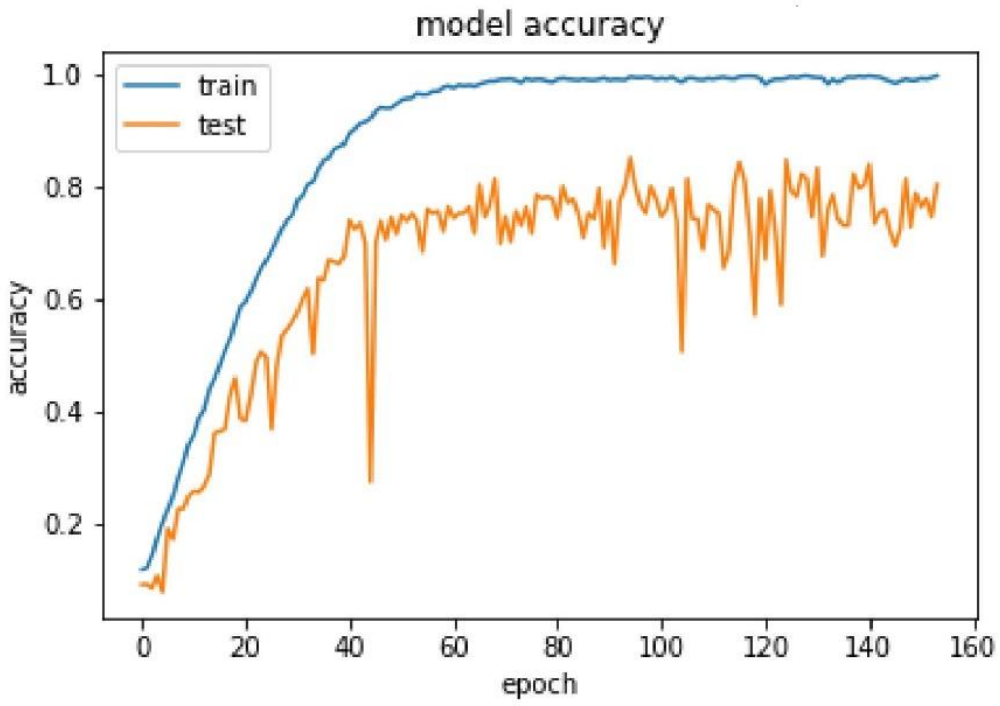

Figure 4.9: Accuracy of the ResNet with 3 blocks

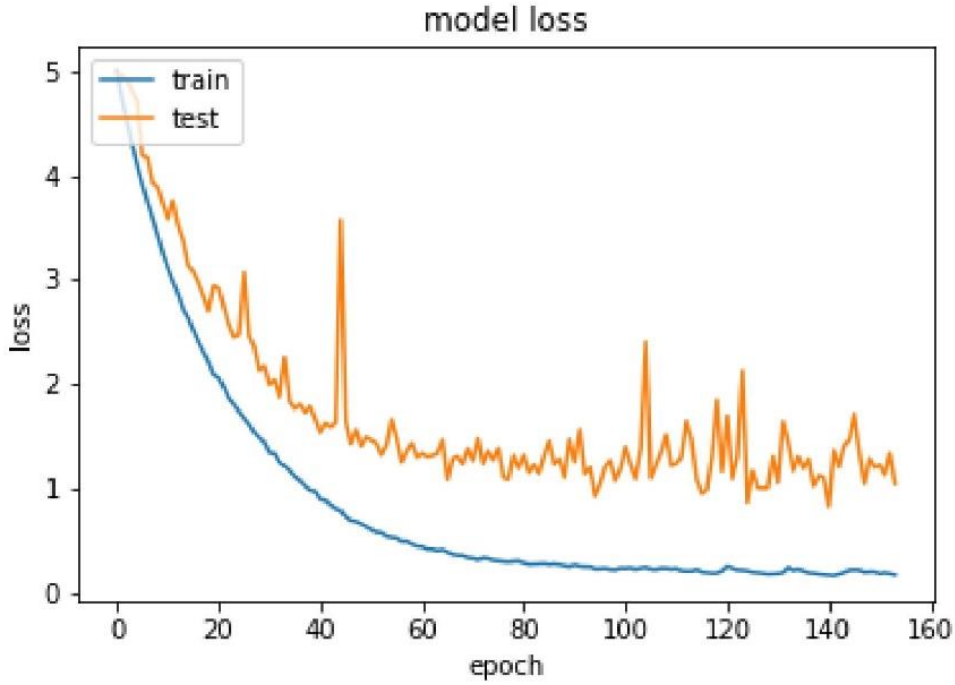

Figure 4.10: Loss of the ResNet with 3 blocks.

From Fig. 28 and Fig. 29, we can see that the accuracy in testing dataset fluctuates around 80%. The model is not so stable. One reason is that the depth of the ResNet is relatively low, only having 7 convolution layers which are used for feature extraction. Another reason is that the number of classes is relatively big. It is equal to the number of people. We have 217 classes. Therefore, it is more complex than some other classification, for example, facial expression recognition which has 7 classes. In fact, we only used the ResNet as a tool for facial feature extraction. By the ResNet, we can transform one image into a vector with 256-dimension – embedding shown in Figure 4.11. In other words, when we want to classify face images, we can use the tool to reduce dimension and extract key features. Then we can implement traditional machine learning techniques such as kNN algorithm to classify face images.

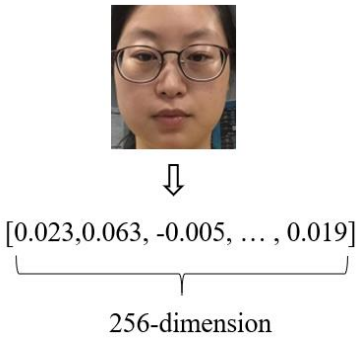

Figure 4.11: A face embedding transformed by the ResNet Model

4.4 Face Classification on Still Images

In this part, part one of Chokepoint dataset is used for face classification. The dataset is a video. The video is recorded when people are walking through a portal one by one. It is similar to the scene of people getting on or off a vehicle. However, frames are randomly extracted to train and test models. These frames are still images. There are 25 people in it. 25 people represent 25 classes. 12 frames or images are extracted for people. 8 images are used to train models and 4 images are used to test models. Thus, there are total 300 images containing 200 training images and 100 testing images.

Then we convert all faces in these frames to face embedding by the pre-trained ResNet model. Two algorithms are implemented for face classification. They are kNN and softmax classifier.

4.4.1 kNN for classification

The most important parameters of a kNN model are the number of neighbors and the metric of distance. The number of neighbors is 5 and the metric of distance is the standard Euclidean metric. We use an object – sklean.neighbors,KNeighborsRegressor from the library scikit-learn to implement the kNN model. The average accuracy is 93% in the testing dataset.

4.4.2 softmax classifier for classification

The softmax classifier is a soft classification. When a face embedding is predicted by a softmax classifier, it yields a probability estimate for each class label. The prediction label is the label with maximum probability. The softmax classifier is usually used as the last layer of CNN. It is implemented by a dense layer from the library Keras. The loss function is categorical cross-entropy. The summary of our softmax classifier is shown in Table 4.3. We train the model with 2000 epochs. The accuracy and loss increase very slowly when the number of epochs arrives at 750. The accuracy of the testing dataset reaches 87%. It is lower than kNN model.

Table 4.3: A summary representation of the softmax classifier

Layer (type)	Output Shape	Param #
dense_1 (Dense)	(None, 25)	6425

Total params: 6,425
Trainable params: 6,425
Non-trainable params: 0

4.5 Face Classification on Real-Time Videos

The Delta dataset is used to train a classification model – kNN. The dataset contains 4 people. Those 4 are labeled as 01, 02, 03 and 04. Then the trained model can classify people when they appear in Pi camera. The distance sensor controls or decides whether face recognition is initialized in the system. When we open the system, the distance sensor continuously detects the distance between the camera and an object. In the system, we set a threshold of distance is equal to 100 cm. When an object is approaching the camera, face recognition is only executed if the distance between the object and the camera is equal to or less than 100 cm. The reason for using the distance sensor is to reduce the working time

of the system and save energy. The motion sensor is to detect whether an object moves. In the system, that whether face recognition can be executed depends on whether the people are moving.

The system can detect strangers when anyone except for those 4 people appears in the Pi camera. The function is met by setting a threshold. In the system, the threshold is set as 0.4. If the distance between a face embedding of a people and its nearest face embedding is less than 0.4, the people are labeled as an “unknown” person. An “unknown” person detected by the system is shown in the Figure 4.12.

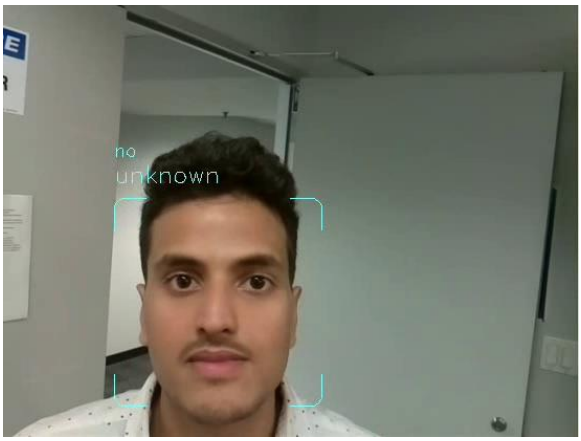

Figure 4.12. An “unknown” person detected by the system

Three experiments are operated for measuring applicable conditions. The first experiment is operated when no distance and motion state is set in the system. In other words, face classification is implemented without the limitations of distance and motion. The second experiment is operated when the distance is set as 100 cm and no motion state is set. When the distance between a people and the system is more than 100 cm, the system does not execute face identification. Whether a person is moving or stationary does not impact the face identification. In the third experiment, face classification is implemented

when the distance between a people and the system is equal to or less than 100 cm and the people must be moving.

In the first experiment, the system real-time identities a face when people are approaching it. The relation between classification accuracy and distance is shown in the Figure 4.13. We can see that the accuracy largely decreases when distance is less than 45 cm or larger than 145 cm. From the Table 4.4, the classification accuracy is good, but detection rate is low. The detection rate is the percentage of frames with detectable faces. It is calculated by the following function:

$$detection\ rate = \frac{the\ number\ of\ frames\ with\ detected\ faces}{the\ total\ number\ of\ frames} \quad (4.9)$$

The average time cost of identifying a face is 0.66 second. The frame rate of the system is 30 fps. The time cost is calculated by the following function.

$$\text{time cost} = \frac{total\ frames}{30\ frames/second} \div \text{frames of detecting faces} \quad (4.10)$$

Table 4.4: Results without distance setting and motion setting

No.	01	02	03	04
Total frames	33	47	46	59
Frames of detecting faces	8	23	6	10
Frames of correct classification	5	21	6	9
Frames of false classification	3	2	0	1
Classification accuracy	63%	91%	100%	90%
Detection rate	24%	49%	13%	17%

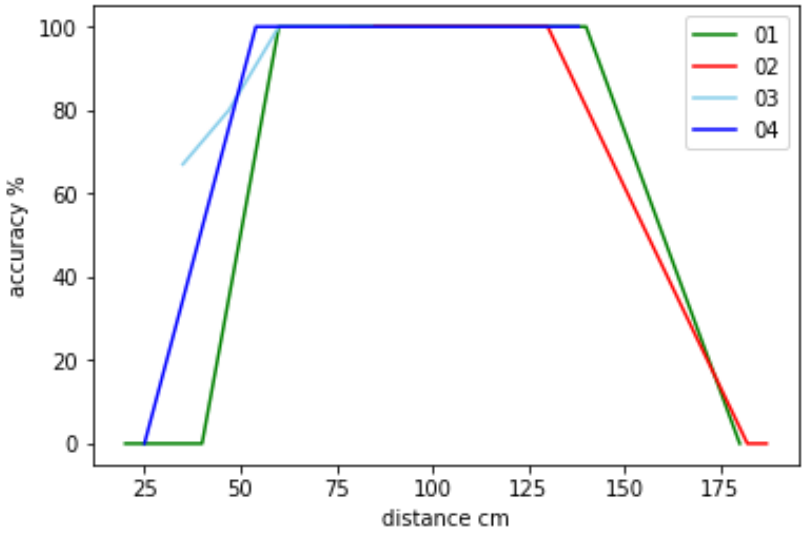

Figure 4.13: The relation between distance and accuracy without limitation of distance and motion

The result of the second experiment is shown in Table 4.5. The classification accuracy is good, and the detection rate is much higher than the first experiment. The average time cost of identifying a face needs 0.19 s.

Table 4.5: Results when the distance is equal to or less than 100 cm and motion is set as False

No.	01	02	03	04
Total frames	30	30	30	30
Frames of detecting faces	24	20	21	19
Frames of correct classification	23	18	19	19
Frames of false classification	1	2	2	0
Classification accuracy	96%	90%	90%	100%
Detection rate	80%	67%	70%	63%

The result of the third experiment is shown in Table 4.6. The classification accuracy has high accuracy. However, the detection rate is very low. It illustrates that detection rate largely decreases when a person is moving. With more limitation of conditions, face detection rate declines. Whether a motion sensor is opened should depend on specific conditions. If people are moving, the system detects the face. If people are static, it does not work. Figure 4.14 and Figure 4.15 show how the system works.

Table 4.6: Results when the distance is equal to or less than 100 cm and motion is set as True

No.	01	02	03	04
Total frames	30	30	30	30
Frames of detecting faces	4	5	4	3
Frames of correct classification	4	5	4	3
Frames of false classification	0	0	0	0
Classification accuracy	100%	100%	100%	100%
Detection rate	13%	17%	13%	10%

Figure 4.14: Face recognition when the distance is less than 100cm and motion is detected

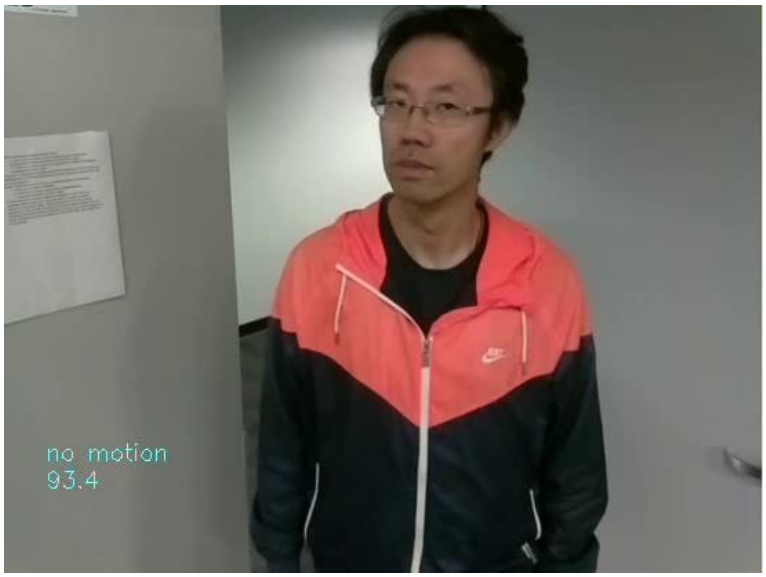

Figure 4.15: No face recognition when the distance is less than 100cm and no motion is detected

CHAPTER V:

CONCLUSION AND FUTURE WORK

5.1 Conclusion

We designed a ResNet with a relatively small number of parameters (573,752) to extract face features. However, some other neural networks in face recognition, for instance, FaceNet, own more than several million to more than one billion parameters. It cannot be denied that they perform well in facial feature extraction. One advantage of our ResNet model is that parameters are much less than these models. Less GPU or CPU memory is needed. It is suitable for implementation in mobile devices, like Raspberry Pi or mobile phone. Our ResNet has 3 blocks. Each block is composed of 2 convolution layers. By comparing the complexity of ResNet, we can see that the model with 3 blocks overperforms models with 4 and 2 blocks. The mechanism of face recognition consists face alignment, facial feature extraction, and face classification. Face alignment is by affine transformation based on 12 points on eyes selected by 68 landmarks. The model of face feature extraction is obtained by training a ResNet with 3 blocks on the LFW dataset. Face classification is to identify faces. By experiments of kNN and softmax classifier, the accuracy of kNN is 93% and the softmax classifier is 87% on recognizing 25 people. In real-time face recognition, distance sensor performs well when people appear 100cm in the front of the system. If distance between people and the system is bigger than 120 cm, the accuracy largely decreases and even it cannot detect people. Motion sensors limit conditions of the system. Therefore, it should be installed according special conditions.

5.2 Future Work

The system needs optimization in future. Several tests and experiments have been left because of lack of time. Future work concerns design of neural network for facial feature extraction, new methods of face classification. This thesis has been mainly

focused on the algorithm of ResNet, the classification model of kNN, real-time processing videos, leaving the study of thresholds of detecting strangers and so on outside the range of the thesis. The following ideas could be made in future:

- The ResNet needs to be optimized because overfitting exists;
- The threshold for detecting strangers needs to tune according to different conditions, and a better method needs to be found for setting thresholds;
- The accuracy of face recognition should be improved.

REFERENCES

[1] “4-year-old dropped off at wrong bus stop on first day of school,” *Washington Post*. [Online]. Available: https://www.washingtonpost.com/video/local/4-year-old-dropped-off-at-wrong-bus-stop-on-first-day-of-school/2016/08/25/c8beee38-6a7c-11e6-91cb-ecb5418830e9_video.html. [Accessed: 14-Nov-2019].

[2] M. D. · C. N. · P. Nov 20, 2016 7:00 AM MT | Last Updated: November 22, and 2016, “Alberta girl, 5, dropped off at wrong bus stop; left alone in the cold | CBC News,” *CBC*, 20-Nov-2016. [Online]. Available: https://www.cbc.ca/news/canada/edmonton/spruce-grove-girl-wrong-stop-chloe-ruffell-1.3858083. [Accessed: 14-Nov-2019].

[3] “Mom sues for $7M after bus drops daughter off at wrong stop.” [Online]. Available: https://nypost.com/2018/06/09/mom-sues-for-7m-after-bus-drops-daughter-off-at-wrong-stop/. [Accessed: 14-Nov-2019].

[4] Nasneen Fathima, P.S. Nivedha, T. Sangavi and S. Selvalakshmi, “Vehicle Tracking System for Children Safety Using RFID, GPS and GSM,” International Journal for Trends in Engineering & Technology, vol. 13, issue 19, May 2016.

[5] “Fact Sheet - Heatstroke Deaths of Children in Vehicles.” [Online]. Available: https://www.noheatstroke.org/original/. [Accessed: 14-Nov-2019].

[6] F. Schroff, D. Kalenichenko, and J. Philbin, “FaceNet: A unified embedding for face recognition and clustering,” in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015, pp. 815–823.

[7] Ming-Hsuan Yang, D. J. Kriegman, and N. Ahuja, “Detecting faces in images: a survey,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 24, no. 1, pp. 34–58, Jan. 2002.

[8] S. Zafeiriou, C. Zhang, and Z. Zhang, “A survey on face detection in the wild: Past,

present and future," *Computer Vision and Image Understanding*, vol. 138, pp. 1–24, Sep. 2015.

[9] P. Viola and M. Jones, "Rapid object detection using a boosted cascade of simple features," in *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*, Kauai, HI, USA, 2001, vol. 1, pp. I-511-I–518.

[10] R. Lienhart and J. Maydt, "An extended set of Haar-like features for rapid object detection," in *Proceedings. International Conference on Image Processing*, 2002, vol. 1, pp. I–I.

[11] K. Cen, "Study of Viola-Jones Real Time Face Detector," p. 5.

[12] T. Ahonen, A. Hadid, and M. Pietikäinen, "Face Recognition with Local Binary Patterns," in *Computer Vision - ECCV 2004*, vol. 3021, T. Pajdla and J. Matas, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 469–481.

[13] K. Kadir, M. K. Kamaruddin, H. Nasir, S. I. Safie, and Z. A. K. Bakti, "A comparative study between LBP and Haar-like features for Face Detection using OpenCV," in *2014 4th International Conference on Engineering Technology and Technopreneuship (ICE2T)*, 2014, pp. 335–339.

[14] V. Kazemi and J. Sullivan, "One millisecond face alignment with an ensemble of regression trees," in *2014 IEEE Conference on Computer Vision and Pattern Recognition*, Columbus, OH, 2014, pp. 1867–1874.

[15] D. E. King, "Max-Margin Object Detection," *arXiv:1502.00046 [cs]*, Jan. 2015.

[16] A. Rosebrock, "Face Alignment with OpenCV and Python," *PyImageSearch*, 22-May-2017.

[17] C. Sagonas, G. Tzimiropoulos, S. Zafeiriou, and M. Pantic, "300 Faces in-the-Wild Challenge: The First Facial Landmark Localization Challenge," in *2013 IEEE*

International Conference on Computer Vision Workshops, Sydney, Australia, 2013, pp. 397–403.

[18] A. Rosebrock, “Facial landmarks with dlib, OpenCV, and Python,” *PyImageSearch*, 03-Apr-2017.

[19] “i·bug - resources - Facial point annotations.” [Online]. Available: https://ibug.doc.ic.ac.uk/resources/facial-point-annotations/. [Accessed: 14-Nov-2019].

[20] I. Ahmad, I. Moon, and S. J. Shin, “Color-to-grayscale algorithms effect on edge detection — A comparative study,” in *2018 International Conference on Electronics, Information, and Communication (ICEIC)*, 2018, pp. 1–4.

[21] M. Turk, and A. Pentland, "Eigenfaces for Recognition," *J. Cognitive Neuroscience,* vol. 3, no. 1, 1991.

[22] F. Rosenblatt, “Two theorems of statistical separability in the perceptron,” in *Proc. Symp.* No. 10 Mechanisation Thought Processes, London, 1959, vol. 1, pp. 421–456.

[23] R. C. Gonzalez, "Deep Convolutional Neural Networks [Lecture Notes]," in *IEEE Signal Processing Magazine*, vol. 35, no. 6, pp. 79-87, Nov. 2018.

[24] D. E. Rumelhart, G. E. Hinton, R. J. Williams, “Learning internal representations by error propagation,” in *Parallel Distributed Processing: Explorations in the Microstructures of Cognition*, vol. 1, D. E. Rumelhart et al., Eds. Cambridge, MA: MIT Press, 1986, pp. 318–362.

[25] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “ImageNet Classification with Deep Convolutional Neural Networks,” in *Advances in Neural Information Processing Systems 25*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2012, pp. 1097–1105.

[26] Y. LeCun, Y. Bengio, and G. Hinton, “Deep learning,” *Nature*, vol. 521, no. 7553,

pp. 436–444, May 2015.

[27] D. Hutchison *et al.*, “Evaluation of Pooling Operations in Convolutional Architectures for Object Recognition,” in *Artificial Neural Networks – ICANN 2010*, vol. 6354, K. Diamantaras, W. Duch, and L. S. Iliadis, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 92–101.

[28] R. Yamashita, M. Nishio, R. K. G. Do, and K. Togashi, “Convolutional neural networks: an overview and application in radiology,” *Insights Imaging*, vol. 9, no. 4, pp. 611–629, Aug. 2018.

[29] J. Kossaifi, Z. C. Lipton, A. Khanna, T. Furlanello, and A. Anandkumar, “Tensor Regression Networks,” *arXiv:1707.08308 [cs]*, Jul. 2018.

[30] “Triplet Loss and Online Triplet Mining in TensorFlow | Olivier Moindrot blog.” [Online]. Available: https://omoindrot.github.io/triplet-loss. [Accessed: 15-Nov-2019].

[31] “davidsandberg/facenet,” *GitHub*. [Online]. Available: https://github.com/davidsandberg/facenet. [Accessed: 15-Nov-2019].

[32] Y. Sun, D. Liang, X. Wang, and X. Tang, “DeepID3: Face Recognition with Very Deep Neural Networks,” *arXiv:1502.00873 [cs]*, Feb. 2015.

[33] Y. Sun, X. Wang, and X. Tang, “Deep Learning Face Representation from Predicting 10,000 Classes,” in *2014 IEEE Conference on Computer Vision and Pattern Recognition*, Columbus, OH, USA, 2014, pp. 1891–1898.

[34] K. He, X. Zhang, S. Ren, and J. Sun, “Deep Residual Learning for Image Recognition,” *arXiv:1512.03385 [cs]*, Dec. 2015.

[35] I. Gruber, M. Hlaváč, M. Železny, A. Karpov, "Facing face recognition with resnet: Round one" in International Conference on Interactive Collaborative Robotics, Springer, pp. 67–74, 2017.

[36] Altman, N. S. “An Introduction to Kernel and Nearest-Neighbor Nonparametric Regression.” *The American Statistician*, vol. 46, no. 3, pp. 175-177,1992.

[37] Kleber, Richard. “The Nature of Statistical Learning Theory,” *The American Mathematical Monthly*, Taylor & Francis Ltd., pp. 616–620, Aug. 1996.

[38] B. H. Boyle, *Support Vector Machines: Data Analysis, Machine Learning and Applications*. Hauppauge, UNITED STATES: Nova Science Publishers, Incorporated, 2011.

[39] “CS231n Convolutional Neural Networks for Visual Recognition.” [Online]. Available: http://cs231n.github.io/linear-classify/#softmax. [Accessed: 15-Nov-2019].

[40] [Online]. Available: http://rinterested.github.io/statistics/softmax.html. [Accessed: 15-Nov-2019].

[41] Scarpino, Matthew. “TensorFlow for Dummies,” *John Wiley & Sons, Incorporated*, 2018.

[42] G. B. Huang, M. Ramesh, T. Berg, and E. Learned-Miller, “Labeled Faces in the Wild: A Database for Studying Face Recognition in Unconstrained Environments,” p. 11.

[43] W. Liu, Y. Wen, Z. Yu, M. Li, B. Raj, and L. Song, “SphereFace: Deep Hypersphere Embedding for Face Recognition,” in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Honolulu, HI, 2017, pp. 6738–6746.

[44] Y. Taigman, M. Yang, M. Ranzato, and L. Wolf, “DeepFace: Closing the Gap to Human-Level Performance in Face Verification,” in *2014 IEEE Conference on Computer Vision and Pattern Recognition*, Columbus, OH, USA, 2014, pp. 1701–1708.

[45] O. M. Parkhi, A. Vedaldi, and A. Zisserman, “Deep Face Recognition,” in

Procedings of the British Machine Vision Conference 2015, Swansea, 2015, pp. 41.1-41.12.

[46] Q. Cao, L. Shen, W. Xie, O. M. Parkhi, and A. Zisserman, "VGGFace2: A Dataset for Recognising Faces across Pose and Age," in *2018 13th IEEE International Conference on Automatic Face & Gesture Recognition (FG 2018)*, Xi'an, 2018, pp. 67–74.

[47] J.-C. Chen, V. M. Patel, and R. Chellappa, "Unconstrained face verification using deep CNN features," in *2016 IEEE Winter Conference on Applications of Computer Vision (WACV)*, 2016, pp. 1–9.

[48] J. Liu, Y. Deng, T. Bai, Z. Wei, and C. Huang, "Targeting Ultimate Accuracy: Face Recognition via Deep Embedding," *arXiv.org*, Jul. 2015.

[49] Y. Wong, S. Chen, S. Mau, C. Sanderson, and B. C. Lovell, "Patch-based probabilistic image quality assessment for face selection and improved video-based face recognition," in *CVPR 2011 WORKSHOPS*, 2011, pp. 74–81.

[50] "ChokePoint Dataset." [Online]. Available: http://arma.sourceforge.net/chokepoint/. [Accessed: 15-Nov-2019].

[51] Kanan, Christopher, et al. "Color-to-Grayscale: Does the Method Matter in Image Recognition? (Color-to-Grayscale: Does the Method Matter?)." *PLoS ONE*, vol. 7, no. 1, Public Library of Science, Jan. 2012, pp. 133-139.